

国际信息工程先进技术译丛

WILEY

虚拟网络——下一代互联网的多元化方法

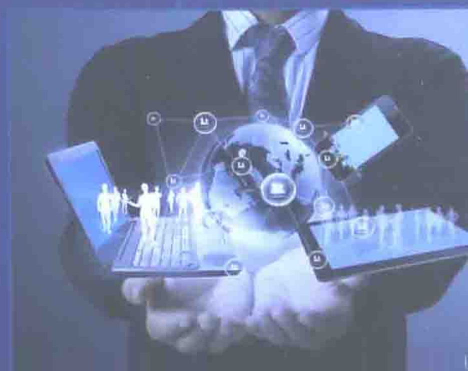
Virtual Networks: Pluralistic Approach for the Next Generation of Internet

[巴西] Otto Carlos M.B. Duarte
[法国] Guy Pujolle 编著

冯玉芬 母景琴 王玲芳 等译









机械工业出版社
CHINA MACHINE PRESS

国际信息工程先进技术译丛

虚拟网络——下一代互联网的多元化方法

［巴西］ Otto Carlos M. B. Duarte
［法国］ Guy Pujolle 编著
冯玉芬　母景琴　王玲芳　等译



机 械 工 业 出 版 社

图书在版编目（CIP）数据

虚拟网络：下一代互联网的多元化方法/（巴西）杜拉特（Duarte, O. C. M. B.），（法国）普杰（Pujolle, G.）编著；冯玉芬等译．—北京：机械工业出版社，2014. 12

（国际信息工程先进技术译丛）

书名原文：Virtual Networks：Pluralistic Approach for the Next Generation of Internet

ISBN 978-7-111-48726-5

Ⅰ. ①虚…　Ⅱ. ①杜…②普…③冯…　Ⅲ. ①虚拟网络　Ⅳ. ①TP393

中国版本图书馆 CIP 数据核字（2014）第 282689 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：张俊红　责任编辑：张俊红
版式设计：霍永明　责任校对：刘怡丹
封面设计：马精明　责任印制：乔　宇
北京机工印刷厂印刷（三河市南杨庄国丰装订厂装订）
2015 年 1 月第 1 版第 1 次印刷
169mm × 239mm · 13. 5 印张 · 267 千字
0 001—2 000 册
标准书号：ISBN 978-7-111-48726-5
定价：69. 80 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
服务咨询热线：(010)88361066	机 工 官 网：www. cmpbook. com
读者购书热线：(010)68326294	机 工 官 博：weibo. com/cmp1952
(010)88379203	教育服务网：www. cmpedu. com
封面无防伪标均为盗版	金 书 网：www. golden-book. com

作为一本专门讲述虚拟网络的科技专著，本书内容涵盖虚拟化技术、两种虚拟化平台及其管理接口，综述了虚拟化联网的现有控制算法，同时描述了使用 Xen 作为虚拟化工具进行报文转发的主要挑战，并详细描述了虚拟网络局部控制的一个建议方案。

本书适合于计算机网络/通信领域的高年级本科生、研究生和研究人员，尤其适合对未来互联网感兴趣的读者。

译 者 序

在不久的将来，互联网将发生巨大变化，为迎接这种变化，信息领域的从业者需要及早介入。虚拟网络作为构建未来互联网的核心技术，在国内外被炒得如火如荼。目前国内在863计划、973计划以及自然科学基金方面对未来互联网都有项目支持，但国外对中国未来互联网的评价是处于摇篮阶段，因此国内急需这方面的内容全面深入的图书和资料。在此背景下，我们推荐引入本书。

就技术的前瞻性而言，本书描述一种多元论的方法，作为后互联网协议（IP）环境的一种新架构，这是未来互联网界的普遍共识之一。本书中给出的多数试验结果来自Horizon项目，这是由法国ANR（Agence Nationale de la Recherche）和巴西Finep（Financiadora de Estudos e Projetos）资助的一个两国共同参与的研究项目。他山之石可以攻玉，希望借此促进国内未来互联网的技术发展。

本书内容涵盖虚拟化技术、两种虚拟化平台及其管理接口，综述了虚拟化联网的现有控制算法，同时描述了使用Xen作为虚拟化工具进行报文转发的主要挑战，并详细描述了虚拟网络局部控制的一个建议方案。本书结构如下：

第1章讨论虚拟化技术，描述Xen、VMware和OpenVZ虚拟化的主要特征并识别出它们的性能折中；第2章详细描述Xen和OpenFlow虚拟化平台，并给出二者的性能分析；第3章讨论在前一章讨论的两种平台的管理工具；第4章描述语境感知的技术和多智能体系统；第5章讨论虚拟化联网的现有控制算法，也分析了使用Xen作为虚拟化工具进行报文转发的主要挑战，并详细描述了虚拟网络局部控制的一种提案；第6章描述引导系统，给出一个多智能体自管理原型；第7章讨论管理和控制功能；第8章详细地描述了用于系统架构的虚拟化技术。

本书由王玲芳负责第1~3章的翻译、全书统稿和校对工作，冯玉芬负责第4~6章的翻译工作，母景琴负责第7~8章的翻译工作。本书在翻译过程中，李虹、潘东升、李冬梅、吴秋义、王弟英、吴璟、游庆珍、李传经、王领弟、王建平等同志参加了部分的翻译工作，在此表示感谢。同时感谢机械工业出版社，感谢出版社的编辑和相关同志。

需要指出的是，本书的内容仅代表原作者个人的观点和见解，并不代表译者及其所在单位的观点。另外，由于翻译时间比较仓促，疏漏、错误之处在所难免，敬请读者原谅和指正。

译者

2015年初于北京

原书前言

当前，研究共同体存在大量积极的工作来重新思考互联网架构，应对互联网当前的限制并支持新的需求。许多研究人员得出结论，对于所有用户和网络提供商的需求而言，不存在均码（one-size-fits-all）的解决方案，因此倡议一种多元论的网络架构。这种新的架构彻底改变了互联网，因为它允许不同协议栈的共存，在同一物理基层上同时运行。因此，本书描述一种多元论的方法，作为后互联网协议（IP）环境的一种新架构。这种后IP架构主要基于带有一个引导系统的虚拟网络，它能够处理各种约束。这种引导系统是面向智能的，并有助于选择最佳参数，通过来自多智能体系统的机制，优化网络的行为。确实，面向自治的架构与网络设备的每部分（路由器、设备等）相关联，这是一种身临其境的方法，将被用来确定语境，并选择和优化控制算法和参数。

本书建议使用的后IP网络的另一个非常重要的概念是网络虚拟化，它将网络抽象为虚拟域（分片/基层）。一个虚拟域代表虚拟路由器而不是物理路由器实例的一个一致的功能组。在这个动态的多栈网络中，多个虚拟网络共存于一个共享的基层之上。这些域将使用引导系统来分配物理资源，并确定哪个虚拟网络将由一名客户使用。在这个语境中，一个服务提供商将能够同时运行具有不同性能和安全等级的多个端到端服务。必要时，可创建和删除虚拟网络。虚拟化支持网络的物理资源更好地加以使用，为客户带来适合的网络。

在本书中给出的多数试验结果来自Horizon项目，这是由法国ANR（Agence Nationale de la Recherche）和巴西Finep（Financiadora de Estudos e Projetos）资助的一个两国共同参与的研究项目。这个国际团体由5个学术和3个业界合作方组成。学术合作方是UPMC—巴黎第6（Laboratoire d'Informatique de Paris 6-LIP6）、Telecom SudParis、Universidade Federal do Rio de Janeiro（UFRJ）、Universidade Estadual de Campinas（Unicamp）和Pontifícia Universidade Católica do Rio de Janeiro（PUC-Rio）。业界合作方是Ginkgo-Networks SA（引导知识平面方面的工作）、Devoteam（研究融合基础设施）和Netcenter Informática LTDA（研究网络设备）。

本书第1章由Luís Henrique M. K. Costa撰写，讨论虚拟化技术，基本上而言，这使我们可共享计算资源，即将一个物理计算环境切片为相互隔离的虚拟计算环境。本章描述了Xen、VMware和OpenVZ虚拟化的主要特征并识别出它们的性能折中。就一台虚拟路由器所用的资源——CPU、RAM内存、硬盘和网络而言，给出虚拟化工具的性能结果。感谢Marcelo Duffles Donato Moreira、Carlo Fragni、Diogo Menezes Ferrazani Mattos和Lyno Henrique Gonçalves Ferraz，是他们定义了基准，并

实施了性能测试。

第2章由 Miguel Elias M. Campista 撰写，详细描述了 Xen 和 OpenFlow 虚拟化平台，并给出了二者的性能分析。选择这两个平台作为 Horizon 项目中所开发新提案的基础。本章也定义了网络虚拟化基础设施必须提供的原语，这就使引导平面可管理虚拟网络单元。感谢 Natalia Castro Fernandes、Marcelo Duffles Donato Moreira、Lyno Henrique Gonçalves Ferraz、Rodrigo de Souza Couto、Hugo Eiji Tibana Carvalho，是他们定义了接口，并实施了试验。

第3章由 Igor M. Moraes 撰写，给出了在前一章讨论的两种平台的管理工具。为了控制和管理网络单元，定义了网络虚拟化基础设施必须提供的5个原语：instantiate（实例化）、delete（删除）、migrate（迁移）、monitor（监测）和 set（设置）。为了验证概念，使用针对两种平台提出的接口，设计和开发了 Xen 平台的一个原型和 OpenFlow 平台的另一个原型。感谢 Diogo Menezes Ferrazani Mattos、Lyno Henrique Gonçalves Ferraz、Pedro Silveira Pisa、Hugo Eiji Tibana Carvalho、Natalia Castro Fernandes、Daniel José da Silva Neto、Leonardo Pais Cardoso、Victor Pereira da Costa、Victor Torres da Costa、Rodrigo de Souza 和 Rafael dos Santos Alves，他们是工具的主要开发人员，并实施了试验。

第4章由 Edmundo R. M. Madeira 和 Guy Pujolle 撰写，描述了语境感知的技术和多智能体系统。引导系统是基于多智能体范型、以一种分布式方式开发的，目的是增加网络的规模扩展性。由此，给出了构造智能体的三个平台。

第5章由 Miguel Elias M. Campista 撰写，讨论虚拟化联网的现有控制算法。本章也分析了使用 Xen 作为虚拟化工具进行报文转发的主要挑战，并详细描述了虚拟网络局部控制的一种提案。在每个物理节点内，这个提案给出了虚拟网络隔离，确保了每个虚拟网络获得的服务水平，即使存在行为不当的虚拟网络的情况下也是如此。在本章描述的称为 XNetMon 的安全虚拟网络监测器，是由 Natalia Castro Fernandes 和 Otto Carlos Muniz Bandeira Duarte 提出和评估的。

第6章由 Edmundo R. M. Madeira 和 Nelson Luís S. da Fonseca 撰写，描述了引导系统。思路是引入一种自治系统，以此处理通信网络日渐增长的复杂性，从处理需要人类干预的任务中解放所需的网络管理员，这些任务如设置管理策略和提升任务的自动化程度——系统配置和优化、灾难恢复和安全。给出了一个多智能体自管理原型。试验是由 Carlos Roberto Senna 和 Daniel Macêdo Batista 实施的。

第7章由 Otto Carlos M. B. Duarte 撰写，讨论管理和控制功能。在监测和得到使用概要之后，知识平面使用预测机制，预测式地检测在虚拟网络配置中更新的必要性。知识平面存储信息，协助管理决策并执行网络维护。模糊控制方案是由 Hugo Eiji Tibana Carvalho 提出并评估的，ADAGA 方案是由 Pedro Silveira Pisa 提出并评估的。

第8章由 Otto Carlos M. B. Duarte 撰写，详细地描述了用于系统架构的虚拟化

技术。基于 Xen 的路由器、OpenFlow 交换机和称为 XenFlow 的这二者的组合体，被用于集成机器和网络虚拟化技术。XenFlow 的关键思路是使用 OpenFlow 管理流，同时也用于支持没有报文丢失条件下的流迁移，并使用 Xen 提供路由和报文转发。XenFlow 是由 Diogo Menezes Ferrazani Mattos 和 Otto Carlos Muniz Bandeira Duarte 提出和评估的。

真诚感谢 Carlos José Pereira de Lucena、Firmo Freire、Djalmal Zeghlache、Jean-François Perrot、Thi-Mai-Trang Nguyen 和 Zahia Guessoum 等教授。也感谢 Marcelo Macedo Achá 和 Cláudio Marcelo Torres de Medeiros。真诚感谢原始思想和文章的葡萄牙的作者们，在本书中没有引用他们的文献，但确实是引入了他们的概念在本书中进行了讨论。最后，也感谢在 Horizon 项目中工作过并给出许多建设性的和深邃评论的所有人：Alessandra Yoko Portella、André Costa Drummond、Andrés Felipe Murillo Piedrahita、Callebe Trindade Gomes、Camila Patrícia Bazílio Nunes、Carlo Fragni、Carlos Roberto Senna、Claudia Susie C. Rodrigues、Cláudio Siqueira Carvalho、Daniel José da Silva Neto、Daniel Macêdo Batista、Diogo Menezes Ferrazani Mattos、Eduardo Rizzo Soares Mendes de Albuquerque、Elder José Reioli Cirilo、Elysio Mendes Nogueira、Esteban Rodriguez Brljevich、Fabian Nicolaas Christiaan van't Hooft、Filipe Pacheco Bueno Muniz Barretto、Gustavo Bittencourt Figueiredo、Gustavo Prado Alkmim、Hugo Eiji Tibana Carvalho、Igor Drummond Alvarenga、Ilhem Fejjari、Ingrid Oliveira de Nunes、Jessica dos Santos Vieira、João Carlos Espiúca Monteiro、João Vitor Torres、Juliana de Santi、Laura Gomes Panzariello、Leonardo Gardel Valverde、Leonardo Pais Cardoso、Luciano Vargas dos Santos、Lucas Henrique Mauricio、Lyno Henrique Gonçalves Ferraz、Marcelo Duffles Donato Moreira、Martin Andreoni Lopez、Milton Aparecido Soares Filho、Natalia Castro Fernandes、Neumar Costa Malheiros、Nilson Carvalho Silva Junior、Othmen Braham、Pedro Cariello Botelho、Pedro Henrique Valverde Guimarães、Pedro Silveira Pisa、Rafael de Oliveira Faria、Rafael dos Santos Alves、Raphael Rocha dos Santos、Renan Araujo Lage、Renato Teixeira Resende da Silva、Ricardo Batista Freitas、Rodrigo de Souza Couto、Sávio Rodrigues Antunes dos Santos Rosa、Sylvain Ductor、Tiago Noronha Ferreira、Tiago Salviano Calmon、Thiago Valentin de Oliveira、Victor Pereira da Costa 和 Victor Torres da Costa。

Otto Carlos M. B Duarte

Guy Pujolle

目　　录

译者序

原书前言

第1章　虚拟化 …… 1

1.1　虚拟化技术 …… 2

1.1.1　完全虚拟化 …… 3

1.1.2　半虚拟化 …… 4

1.2　虚拟化工具 …… 4

1.2.1　Xen …… 4

1.2.2　VMware …… 6

1.2.3　OpenVZ …… 9

1.3　场景和方法论 …… 10

1.4　性能评估 …… 12

1.4.1　CPU 性能 …… 13

1.4.2　内存性能 …… 13

1.4.3　硬盘和文件系统性能 …… 13

1.4.4　网络性能 …… 14

1.4.5　整体性能——Linux 内核编译 …… 14

1.4.6　单个虚拟机测试 …… 14

1.4.7　多虚拟机测试 …… 19

1.5　小结 …… 25

1.6　参考文献 …… 26

第2章　虚拟网络接口 …… 27

2.1　虚拟网络：隔离、性能和趋势 …… 28

2.1.1　网络虚拟化方法 …… 28

2.1.2　网络虚拟化技术 …… 30

2.1.3　Xen 和 OpenFlow 网络虚拟化技术的特征 …… 34

2.1.4　性能评估 …… 40

2.2　Xen 原型 …… 47

2.2.1　虚拟机服务器 …… 48

2.2.2　虚拟机服务器客户端 …… 49

2.2.3　图形用户界面 …… 51

2.3 OpenFlow 原型 …… 52
2.3.1 应用 …… 52
2.3.2 OpenFlow Web 服务器 …… 53
2.3.3 图形用户界面 …… 55
2.4 小结 …… 56
2.5 参考文献 …… 56

第3章 虚拟网元的性能改进和控制 …… 59
3.1 基于 Xen 的原型 …… 60
3.1.1 Xen 迁移 …… 61
3.1.2 Xen 统计信息 …… 64
3.1.3 Xen 拓扑 …… 65
3.1.4 虚拟化硬件改进 …… 66
3.2 基于 OpenFlow 的原型 …… 67
3.2.1 FlowVisor …… 68
3.2.2 OpenFlow 迁移 …… 70
3.2.3 OpenFlow 统计 …… 71
3.2.4 OpenFlow 发现 …… 71
3.2.5 OpenFlow 生成树 …… 73
3.3 小结 …… 75
3.4 参考文献 …… 75

第4章 语境感知技术的最新状态 …… 78
4.1 自治系统 …… 78
4.1.1 自治系统的特点 …… 78
4.1.2 自治系统的架构和操作 …… 79
4.2 采用多代理系统进行引导 …… 81
4.2.1 代理的定义 …… 81
4.2.2 代理的特点 …… 81
4.2.3 认知代理 …… 82
4.2.4 反应式代理 …… 82
4.2.5 多代理系统 …… 82
4.3 构建自治平台的选项 …… 83
4.3.1 Ginkgo …… 84
4.3.2 DimaX …… 85
4.3.3 JADE …… 87
4.4 网络控制的语境感知技术 …… 90
4.4.1 语境感知系统架构 …… 91

4.4.2 感知子系统 …… 92
4.4.3 思考子系统 …… 94
4.4.4 动作子系统 …… 96
4.5 小结 …… 99
4.6 致谢 …… 99
4.7 参考文献 …… 99

第5章 向虚拟网络提供隔离和服务质量 …… 102
5.1 虚拟网络控制和管理背景知识 …… 102
5.2 使用 Xen 进行报文转发中的挑战 …… 104
5.3 控制域0共享的资源 …… 106
5.4 小结 …… 112
5.5 参考文献 …… 112

第6章 引导系统 …… 114
6.1 自治引导系统 …… 114
6.1.1 架构 …… 115
6.1.2 Horizon 项目的引导平面 …… 116
6.1.3 相关工作 …… 117
6.1.4 引导、管理和虚拟化平面的相互作用 …… 118
6.1.5 在 Horizon 架构中引导平面的职责 …… 118
6.2 引导平面功能和需求 …… 119
6.3 初步引导平面设计 …… 119
6.3.1 动态规划器 …… 121
6.3.2 行为 …… 122
6.3.3 系统内和系统间视图 …… 128
6.3.4 APS 的接口 …… 128
6.4 引导代理 …… 130
6.5 测试床 …… 132
6.5.1 工具 …… 133
6.5.2 测试床中的试验 …… 135
6.6 多代理 APS …… 136
6.7 结果 …… 138
6.8 用于虚拟网络自管理的多代理系统 …… 140
6.8.1 原型实现 …… 140
6.8.2 试验结果 …… 141
6.9 小结 …… 146
6.10 参考文献 …… 147

第 7 章 管理和控制：共置视图 …… 150
7.1 动态 SLA 控制器 …… 151
7.1.1 有关虚拟网络 QoS 的背景知识 …… 151
7.1.2 建议的模糊控制系统 …… 152
7.1.3 结果 …… 157
7.2 局部信息的更新预测机制 …… 160
7.2.1 异常检测系统的背景 …… 160
7.2.2 ADAGA 系统 …… 161
7.2.3 异常系统评估 …… 165
7.3 小结 …… 169
7.4 参考文献 …… 170

第 8 章 系统架构设计 …… 173
8.1 整体架构设计 …… 174
8.1.1 Xen 架构 …… 174
8.1.2 OpenFlow 管理架构 …… 186
8.2 一个混合的 Xen 和 OpenFlow 系统架构设计 …… 189
8.2.1 Xen 和 OpenFlow 虚拟化平台的优势和劣势 …… 191
8.2.2 XenFlow 架构设计 …… 192
8.2.3 试验结果 …… 196
8.3 小结 …… 198
8.4 参考文献 …… 199

附录 英文缩略语释义对照表 …… 202

第1章 虚 拟 化

在本书中，将焦点放在基于多元化方法的一种新颖互联网架构上。一个多元化架构的例子如图1.1所示。在图1.1中，每个路由器层代表具有独立协议栈的一个不同网络，它们共享来自底层处基础网络设施的各种资源。虚拟化是使这样一种多元化架构成为可能的一项关键技术。虚拟化是这样一项技术，基本上说，它支持计算资源的共享［POP 74］。虚拟化将一个真实计算环境分成虚拟计算环境，这些环境是相互隔离的，并按照从真实的非虚拟化的环境中所期望的那样，与上面的计算层交互。一个虚拟化环境和一个非虚拟化环境之间的比较如图1.2所示。图的左手侧给出一个传统的计算环境，其中各项应用是在一个操作系统（OS）之上执行的，操作系统控制基础硬件。在图的右手侧，给出一个虚拟化环境，其中一个虚拟化层支持多个OS并行运行，每个OS都有其自己的应用，并控制它们到硬件的访问。当处理虚拟网络时，考虑路由器资源，如处理器、内存、硬盘、队列和带宽，这和计算环境虚拟化时是一样的。一个虚拟路由器和链路集合被称作一个虚拟网络。因此，使用虚拟化技术，可有多个并行的虚拟网络，每个网络都有一个特定的网络协议栈，共享单个物理网络基础设施，如图1.1所示。

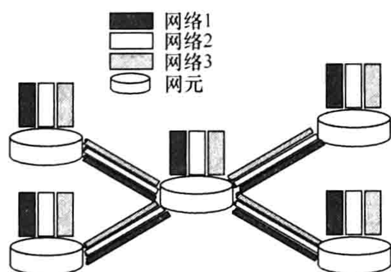


图1.1 多元化架构范例



图1.2 虚拟化环境范例

虚拟化普遍实现为称为hypervisor的一个软件层，它负责在多个虚拟环境或虚拟机（Virtual Machine，VM）之间复用计算资源。每个VM运行在hypervisor之上，由hypervisor控制到物理资源的访问。存在不同的hypervisor和虚拟化技术。本章给出最普遍的虚拟化工具（Xen［BAR 03，CHI 08］、VMware［VMW 07a］和Open-VZ［KOL 06］）的主要特征概述并识别性能折中问题。就一台虚拟路由器的所关注资源方面［中央处理单元（CPU）、随机读取内存（RAM）内存、硬盘和网络］，该项研究比较了虚拟化工具的性能。虚拟路由器使用CPU处理到达报文，并依据转发表路由报文。使用RAM存储转发表。硬盘的主要用途是存储VM映像。使用

网络资源转发报文，这是一台路由器的主要任务。对于虚拟路由器的正常操作，CPU、RAM 和网络是虚拟化额外负担的最敏感资源。磁盘性能额外负担是人们所关注的，原因是它影响新路由器的实例化和虚拟路由器的迁移。为更好地理解由这种工具引入的额外负担，只要可能的情况，也给出原生的性能。

因为虚拟化可导致时间敏感应用的误操作（malfunction）[VMW 08]，所以使用与相关工作中所用技术 [XEN 07，XMW 07b] 不同的一种技术。将结果基于这样的时间，即一个虚拟化系统完成一项任务所需的时间，是从一个外部非虚拟化计算机测量得到的。

实施了两种类型的试验。第一种类型的试验目标是分析由额外层诱发的性能损失（hypervisor）是由虚拟化工具引入的。为取得这个目标，在第一种类型的试验中，仅有一个 VM 运行在虚拟化软件之上。将原生 Linux 的性能与 Xen、VMware 和 OpenVZ 虚拟化软件的性能进行了比较。结果表明，Xen 是基于 PC 的路由器虚拟化的一种良好适用物（fit），具有可接受的虚拟化额外负担（这在 1.4.6 节做了展示），支持 hypervisor 修改，是开源的，并提供虚拟路由器灵活性，原因是它具有一个虚拟硬件接口，支持在不同虚拟路由器中使用不同的 OS。另外，单个 VM 试验为第二种类型的试验（处理多个 VM）提供一个基线。第二个试验集合深入探讨所选中的虚拟化工具如何随着并行运行的 VM 数量而规模扩展的。这些种类的测试有如下目标，即澄清实例化的多个 VM（消耗同样的资源）如何影响总体性能，且虚拟化工具如何处理公平性。在这种类型的测试中，也验证各 VM 间 CPU 核分配的不同方案如何影响总体性能。

在 1.1 节，给出测试虚拟化工具所用的技术。在 1.2 节，详细描述虚拟化工具。1.3 节描述测试方法论，并给出测试床。1.4 节给出虚拟化工具比较中使用的基准测试，1.4.6 节和 1.4.7 节给出性能比较结果。最后，在 1.5 节给出本章的结语。

1.1 虚拟化技术

为了更好地理解所比较的虚拟化工具，重要的是识别不同的虚拟化技术。本节描述 Xen、VMware 和 OpenVZ 使用的概念和技术。

虚拟化有许多不同的定义，但就虚拟化是共享计算资源、在虚拟环境之间赋予一定程度的隔离的一项技术方面，这些定义是一致的。依据 Popek 和 Goldberg [POP 74]，经典定义是，虚拟化是提供 VM 的一项技术，这些 VM 是底层硬件的高效被隔离复制。如今，这个概念不仅可在硬件上一般化，而且可在任何计算资源上一般化，作为一个 OS 内核或编程语言（如 Java 和 C#）使用的 VM 抽象。

从虚拟目标中出现了几项挑战。第一个挑战是调用所有虚拟环境访问相同的底层计算资源。对于硬件虚拟化，共享资源是 CPU、RAM、存储和网络。RAM 共享

访问可以不同方式完成。虚拟环境可被赋予到虚拟内存空间（可被转换为物理RAM，这和OS的做法相同）的访问权限。另一种方法是让VM知道它们的虚拟化特征，并允许它们在由hypervisor指派一个区域内存片之后，直接访问内存。CPU共享可以几种方式完成，并可使用诸如轮转法、加权的轮转法、应需分配和其他方法等机制做到。一般而言，输入/输出（I/O）可使用存储交换数据的缓冲这样一种统一的方式加以处理，这些数据是在物理外设和虚拟外设之间复用和解复用的。

试验使用商用的基于x86的硬件，这对虚拟化施加额外挑战。20世纪70年代，在虚拟化开发的开始阶段，所设计硬件都是支持虚拟化的。大型主机有这样的指令集，其中处理资源分配和使用的所有指令都是特权指令，即程序要求一定的CPU特权执行等级。在这些CPU架构中，可使用称作“去特权”[ADA 06]法的一项技术做到硬件虚拟化，其中虚拟化OS是在一个非特权语境中执行的，目的是无论何时资源分配或使用指令将被执行时，就产生陷阱。出于这个原因，hypervisor将截获陷阱，并以一种对其他VM安全的方式来模拟所需资源的分配或使用。依据Popek和Goldberg[POP 74]的说法，为硬件虚拟化构造一个hypervisor存在三个需求：①效率，这意味着来自虚拟CPU（vCPU）之指令集的一个大型子组应该直接在真实CPU中执行，不需要来自hypervisor的任何干预；②资源控制，这意味着hypervisor必须对所有资源具有完全的控制；③等价性，这意味着hypervisor必须向虚拟环境提供等价于原始接口的一个虚拟接口。20世纪70年代的大型主机将有利于构造hypervisor，原因是使用去特权技术就可做到VM隔离。对于基于x86的硬件，情况就不是这样的，原因是出于优化目的，基于x86的硬件指令集具有访问共享资源的指令，但不要求一个特权化的语境。此外，基于x86的指令集包含这样的一组指令，它们被归类为对特权等级是敏感的，其中指令以取决于当前特权等级的一种不同方式执行。如果一个去特权的OS执行一条敏感的指令，它将悄无声息地失败，原因是它不会产生一条陷阱，而且它也不以OS所拟设的方式执行。为处理这些基于x86的硬件问题，存在几项解决方法，在描述不同虚拟化技术的后面各节中将给出。

1.1.1 完全虚拟化

完全虚拟化是这样一种虚拟化技术，其中虚拟化所有的原始接口，且输出给虚拟环境的各接口与原始接口完全相同。在这种方法中，guest OS（寄居OS），即驻留在VM内的OS，不需要修改，直接在VM内执行。为处理来自基于x86硬件平台的敏感指令，可使用不同技术。一种著名的技术是二进制翻译。二进制翻译检查要执行的代码，搜索存在问题的指令，并将它们替换为模拟期望性能的那些指令。二进制翻译的优势是，它支持应用和OS在不做修改的条件下加以使用。不过，二进制翻译诱发高的CPU额外负担，因为所有执行的代码都必须做检查，且存在问题的指令必须在运行时加以替换。

最近，存在来自主要硬件制造商的大力投入来优化虚拟化。服务器合并法使用虚拟化将具有空闲容量的几台服务器替换到单个硬件，该硬件具有较高的利用率，其中执行几个虚拟服务器。服务器合并法已经成为共性实践，它在主要公司中削减设备和维护成本。出于这个原因，AMD 和 Intel 都为在现代 CPU 中提供更高效的虚拟化支持开发了各项技术。Intel 虚拟化技术（IVT）和 AMD 虚拟化（AMD-V）都为完全虚拟化提供较佳的性能，方法是引入两种新的操作模式：root 和 non-root。root 操作模式由 hypervisor 使用，类似于常规 CPU 操作，提供完全的 CPU 控制和传统的特权等级的四个环。non-root 模式是用于 VM 的执行的。在这种模式中，CPU 也提供特权等级的四个环，寄居 OS 不再在一个去特权化环中执行，而是在环 0 中执行，这个环就是为之设计的。无论何时寄居 OS 执行一条存在问题的指令，CPU 就产生一个陷阱，并将控制返给 hypervisor，来处理这条指令。采用这种 CPU 支持，二进制翻译就不再是必要的，且完全虚拟化 hypervisor 就极大地增加了它们的性能。

1.1.2 半虚拟化

半虚拟化是这样一种虚拟化技术，其中寄居 OS 与 hypervisor 协作得到较佳的性能。在半虚拟化中，修改寄居 OS，无论何时执行一条存在问题的指令，就调用 hypervisor。敏感指令被替换为调用 hypervisor 的一条虚拟化感知指令。出于这个原因，hypervisor 不需要为存在问题的指令而监测 VM 执行，相比于使用二进制翻译的完全虚拟化，这极大地降低了额外负担。折中代价是，OS 必须被修改和重新编译，产生半虚拟化的 OS 映像。为进行 hypervisor 调用，需要半虚拟化的 OS 映像。这阻碍了遗留 OS 的虚拟化，并要求 OS 开发商的协作才能完成。

1.2 虚拟化工具

本节描述主要的虚拟化技术，并给出 Xen、VMware 和 OpenVZ 的性能评估结果。

1.2.1 Xen

Xen 是一个开源 hypervisor，提出它，为的是运行在商用硬件平台上，它使用半虚拟化技术 [BAR 03]。Xen 使我们可在单个物理机器上同时运行多个 VM。Xen 架构由一个 hypervisor（位于物理硬件之上）和 hypervisor 之上的几个 VM 组成，如图 1.3 所示。每个 VM 可有其自己的 OS 和应用。hypervisor 控制到硬件的访问，也管理由各 VM 共享的可用资源。另外，为提供可靠的和高效的硬件支持 [EGI 07]，设备驱动被放在一个孤立的 VM [称作 Domain 0（dom0）] 中。因为 dom0 具有对物理机器硬件的完整访问权限，所以相比其他 VM，它有特权，参见用户域（domUs）。另外，domUs 有虚拟驱动，称作前台驱动，它与位于 dom0 中的后台驱

动通信来访问物理硬件。接下来简短地解释 Xen 如何将所关注的每种机器资源（处理器、内存和 I/O 设备）虚拟化为一台虚拟路由器。

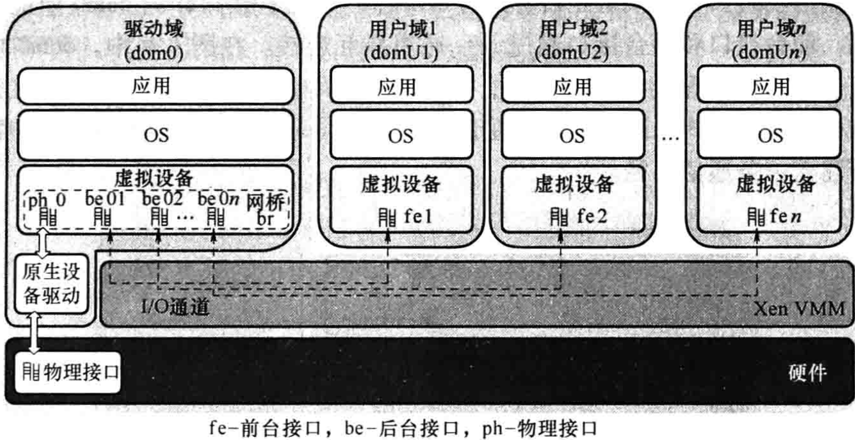


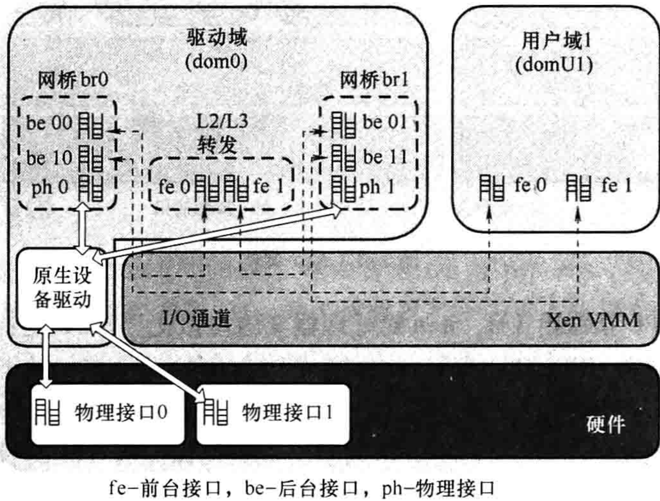
图 1.3 Xen 架构

通过将 vCPU 指派给 VM，Xen 对处理器实施虚拟化。vCPU 是在每个 VM 内运行进程可看到的各 CPU。Xen hypervisor 实现一个 CPU 调度器，动态地将一个物理 CPU 在某个时段期间映射到每个 vCPU。Xen 所使用版本（3.2）的默认调度器是信用调度器，实施一个比例性的 CPU 共享。依据指派给各 VM 的权重，信用调度器将 CPU 资源分配给每个 VM（或更具体而言，是分配给每个 vCPU）。信用调度器在对称多处理（SMP）主机上也可以是工作守恒的。这意味着调度器允许物理 CPU 运行在 100% 处，如果任何 VM 有工作要做。在一个工作守恒的调度器中，对一个 VM 可使用的 CPU 资源量是没有限制的。

在 Xen 中的内存分配目前是以静态方式完成的。每个 VM 接收定量的内存空间，这是在其创建时刻被指定的。另外，为要求来自 hypervisor 的最小干预，各 VM 负责分配和管理硬件页表的相应部分。当一个 VM 每次要求一个新的页表时，它就从其自己的内存空间中分配并初始化一个页，且将之注册到 Xen hypervisor，后者负责确保隔离性。

在 Xen 中，使用共享内存异步缓冲描述符环，来自 I/O 设备的数据被传递进出每个 VM。Xen hypervisor 的任务是实施确认检查。例如，检查各缓冲被包含在一个 VM 内存空间内。通过使用它的原生设备驱动，dom0 直接访问 I/O 设备，同时代表 domUs 实施 I/O 操作。另外，domUs 使用其后台驱动从 dom0 请求设备访问 [MEN 06]。一种特殊情形的 I/O 虚拟化是网络虚拟化，它负责将来自物理接口的到达报文解复用到各 VM，同时复用由各 VM 产生的外发报文。图 1.4 所示为 Xen 使用的默认网络架构。对于每个 domU，Xen 创建由这个 domU 要求的虚拟网络接口。这些接口也称作前台接口，domUs 用之进行其所有的网络通信。此外，在

dom0 中创建后台接口，对应于在一个 domU 中的每个前台接口。为在后台和前台接口之间交换数据，Xen 使用一个 I/O 通道，它使用一种零复制机制。这种机制将包含数据的物理页重新映射到目标域［MEN 06］。后台接口作为 dom0 中虚拟接口的代理。前台接口和后台接口通过 I/O 通道相互连接。在图 1.4 中，dom0 中的后台接口被连接到物理接口，同时通过一个虚拟网桥相互连接。Xen 使用的这种默认架构被称作网桥模式。因此，I/O 通道和网桥在 domUs 中创建的虚拟接口和物理接口之间建立一条通信路径。



fe-前台接口，be-后台接口，ph-物理接口

图 1.4　Xen 网络架构

1.2.2　VMware

VMware 是为端用户和数据中心顾客提供机器虚拟化平台的一家公司。VMware 虚拟化平台是基于全虚拟化概念的。本项工作评估称作 VMware ESX Server 的一个 VMware 数据中心类虚拟化平台。它主要用于服务器合并法，且是企业虚拟化最常用组件之一。VMware ESX Server 目标是基于由系统管理员设置的资源分配策略，保障 VM 隔离和资源共享公平性。资源共享是动态的，原因是可应需地将资源分配和重新分配给各 VM［VMW 05］。虽然这项工作评估 VMware ESX Server 3.5，但这里报告的信息部分是来自 VMware ESX Server 2.5 的，这源于如下事实，即 VMware ESX Server 是一项专用产品，且就其实现方面几乎没有多少信息。下面的描述考虑的是版本 2.5，我们认为在版本 2.5 和 3.5 之间没有显著的改变。

如图 1.5 所示，VMware 架构由硬件接口组件、虚拟机监测器（VMM）、VMkernel、资源管理器和服务控制台组成。硬件接口组件负责实现硬件特定的功能，并创建要提供给各 VM 的一个硬件抽象。它使各 VM 的硬件是独立的。VMM 负责 CPU 虚拟化，向每个 VM 提供一个 vCPU。VMkernel 控制和管理硬件基层。

VMM 和 VMkernel 一起实现虚拟化层。资源管理器是由 VMkernel 实现的。它将底层物理资源在 VM 间进行分割，为每个 VM 分配资源。VMkernel 也实现硬件接口组件。服务控制台实现各种服务（如启动、发起虚拟化层和资源管理器的执行），并运行各项应用，它们实现支持、管理和后勤服务功能。

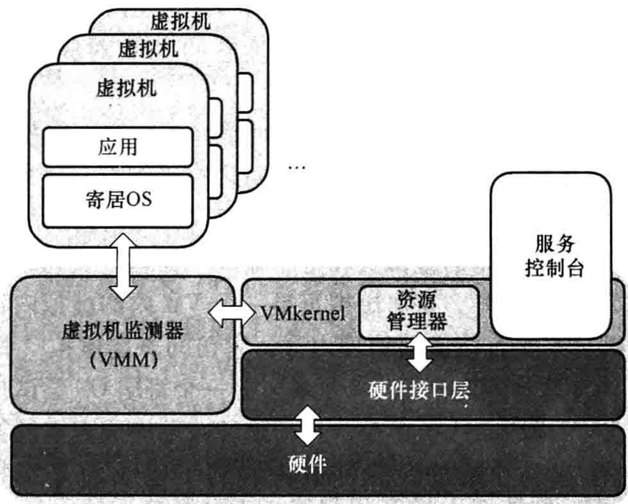


图 1.5 VMware 架构

与其他虚拟化工具一样，VMware ESX Server 虚拟化四种主要资源：CPU、内存、磁盘和网络设备。下面将进一步详细描述 VMware 是如何虚拟化每种资源的。

通过为每个 VM 设置一个 vCPU，完成 CPU 虚拟化。VM 不会认识到它正运行在一个 vCPU 之上，原因是 vCPU 似乎都有其自己的寄存器和控制结构［VMW 05］。一个 VM 可有一个或两个 vCPU。当它有一个以上的 CPU 时，称作一个 SMP VM。VMM 负责 CPU 虚拟化，方法是设置系统状态，并执行由 VM 发出的指令。

在一个虚拟化环境中，与其所设计的相比，寄居 OS 运行在一个低的特权等级。如在 1.1 节所述，虚拟化 CPU 资源的一种经典方法是陷阱和模拟（trap-and-emulate），这是这样的一种技术，其中 VM 尝试执行一条指令，如果它不能在一个低的特权等级被执行，则 CPU 产生一个要由 VMM 处理的陷阱，VMM 为寄居 OS 模拟指令执行。不过，如在 1.1 节所述，对于 x86 架构，这项技术是不工作的，原因是它有对特权等级敏感的指令，且以与 OS 所拟设的一种不同方式执行，而不会产生一个陷阱。为解决这个问题并保持一个令人满意的性能，VMware 组合使用两种 CPU 虚拟化技术：直接执行和 CPU 模拟。来自一个 VM 的用户空间的指令，直接在物理 CPU 上执行，这是称作直接执行的一项技术。对特权等级敏感的指令被 VMM 捕获，VMM 模拟指令执行，这增加了性能额外负担。组合使用两种技术的方法，使 CPU 密集的用户空间应用具有接近原生系统的性能。性能损失取决于必须要被替换的敏感指令数量。

CPU 调度是由资源管理器完成的。CPU 调度依据的是份额，这是用来度量要将多少时间赋予每个 VM 的单位。CPU 调度是比例性份额的，这意味着赋予每个 VM 的 CPU 时间正比于它拥有的份额量占系统中份额总量的多少。在一个 SMP VM 中，CPU 分配是不同的。资源管理器将 vCPU 一个一个地调度到物理 CPU 上，并尝试同时执行它们。VMware CPU 调度尝试在各 VM 的 CPU 分配之间保持公平性。当一个 VM 被中止或空闲时，资源管理器调度它的 CPU 时间到正在运行的其他 VM 上。

VMware 的内存虚拟化方法是创建内存地址翻译的一个新层次。它是这样完成的，即向每个寄居 OS 提供一个虚拟页表，这对内存管理单元（MMU）［BAR 03］是不可见的。在一个 VMware 虚拟化环境内，寄居 OS 访问提供给 VM 的一个虚拟内存空间。寄居 OS 页表在寄居虚拟页和寄居虚拟“物理”页间维持一致性。与在一个原生 OS 虚拟内存中一样，寄居虚拟页是一个 VM 内的虚拟内存页。但是，一种寄居虚拟页面机制不能直接访问物理内存，它访问寄居虚拟“物理”内存。寄居虚拟“物理”内存是物理内存的一个抽象。当一个寄居 OS 尝试执行访问物理内存的一条指令时，VMM 捕获这条指令，且其地址被翻译成真实的物理地址。寄居虚拟“物理”内存总是连续的，但可被映射到不连续的真实物理内存之上。VMware内存共享遵循管理策略，它们（例如）定义要由一个 VM 访问的最小和最大物理内存量。也可能出现这种情况，即各 VM 消耗的量大于物理机器上可用的物理内存总量。这是可能的，因为主机系统也可进行交换，这就像在现代 OS 中所用的传统虚拟内存机制一样。内存共享调度器就像 CPU 调度器一样工作，但考虑内存份额而不是 CPU 份额。

VMware 的 I/O 虚拟化方法是模拟性能关键的设备，如磁盘和网络接口卡。设备访问由 VMkernel 模拟。VMkernel 调用硬件接口层，它负责访问设备驱动，并在物理硬件设备上执行操作。对于存储虚拟化，向 VM 提供一个小型计算机系统接口（SCSI）驱动。各 VM 访问这个驱动，且 VMkernel 捕获驱动访问指令，并像主机文件系统中的文件一样实现 VM 磁盘。

就网络 I/O 虚拟化而言，VMware 实现 vmxnet［VMW 05］设备驱动，它是底层物理设备的一个抽象。当一个应用希望通过网络发送数据时，寄居 OS 处理请求，并调用 vmxnet 设备驱动。I/O 请求被 VMM 截获，且控制被传递给 VMkernel。VMkernel 独立于物理设备。它处理请求，管理各种 VM 请求，并调用硬件接口层，它实现特定的设备驱动。当数据到达物理接口时，将之发送到特定 VM 的机制是相同的，但顺序相反。由这种机制引入的主要额外负担是 VM 和 VMkernel 之间的语境切换。为减少由语境切换导致的额外负担，在进行一次语境迁移之前，VMware ESX Server 收集发送或接收网络报文的分簇（clusters）。仅当报文速率高到足够可避免增加的报文时延时，才使用这种机制。

可做个结论，即 VMware ESX Server 是一个全虚拟化工具，它提供许多个管理

和后勤操作工具。VMware ESX Server 将焦点放在数据中心虚拟化上。它提供一个灵活的、高性能的 CPU 和内存虚拟化。但是，I/O 虚拟化仍然是一个问题，因为它是通过模拟物理设备完成的，且涉及语境改变。

1.2.3 OpenVZ

OpenVZ 是一个开源的 OS 级虚拟化工具。OpenVZ 支持在单个 OS 内核之上的多个隔离的执行环境。每个被隔离的执行环境被称作一个虚拟专用服务器（VPS）。一个 VPS 看起来就像一台物理服务器，拥有其自己的进程、用户、文件、互联网协议（IP）地址、系统配置，并提供全部的 root shell 访问权限。OpenVZ 宣称为引入较少额外负担的虚拟化工具，因为每个 VPS 共享相同的 OS 内核，提供一个高级的虚拟化抽象。这个虚拟化技术的主要用途是 web 托管（为每名顾客提供一个完备的 Linux 环境）和信息技术（IT）教育机构（为每名学生提供一台 Linux 服务器，该机器可被远程地监测和管理 [SWS 05]）。除了 OpenVZ 引入的较小额外负担外，相比其他虚拟化工具（如 VMware 或 Xen），它是不太灵活的，因为 OpenVZ 执行环境必须是基于物理服务器的相同 OS 内核上的一个 Linux 发行版。

如图 1.6 所示的 OpenVZ 架构由运行在硬件之上的一个修改过的 Linux 内核组成。OpenVZ 修改过的内核实现几个子系统的虚拟化和隔离、资源管理和检查点 [KOL 06]。另外，I/O 虚拟化机制是由 OpenVZ 修改过的内核提供的，它拥有每个 I/O 设备的设备驱动。这个修改过的内核也实现一个两层进程调度，在第一层进程调度，负责定义哪个 VPS 将运行；在第二层进程调度，负责决定哪个 VPS 进程将运行。两层调度器和在各 VPS 之间提供隔离的一些特征，形成 OpenVZ 虚拟化层。各 VPS 运行在 OpenVZ 虚拟化层以上。每个 VPS 有其自己的应用和软件包集合，这些是包含应用或服务的某些 Linux 发布版的分裂形式（segmentation）。因此，一个 VPS 有其自己的服务和应用，它们是相互独立的。

通过允许或禁止一个 VPS 访问物理服务器上的一个资源，就可在 OpenVZ 中完成资源虚拟化。一般而言，在 OpenVZ 中的资源不是模拟的，它们是在 VPS 间共享的。为定义每项资源的量（针对每个 VPS 要加以保障），在 VPS 配置文件中定义了许多计数器（大约 20 个）。接下来进一步详细描述 OpenVZ 如何虚拟化处理器、内存、磁盘和网络设备。

对于处理器虚拟化，OpenVZ 实现一个两级 CPU 调度器 [KOL 06]。在第一级，虚拟化层确定在每个时间片将执行哪个 VPS，这考虑到 VPS CPU 优先级，是以 cpuunits（cpu 单元数）度量的。在第二级，是运行在 VPS 内部的，标准 Linux 调度器在每个时间片将执行哪个进程，这考虑到标准进程优先级参数。

OpenVZ 允许各 VPS 直接访问内存。此外，它比其他虚拟化技术（如 Xen）要灵活。在 VPS 执行过程中，专用于一个 VPS 的内存量可由主机管理员进行动态改变。OpenVZ 内核管理 VPS 内存空间，使之保持在物理内存中，这是对应于运行

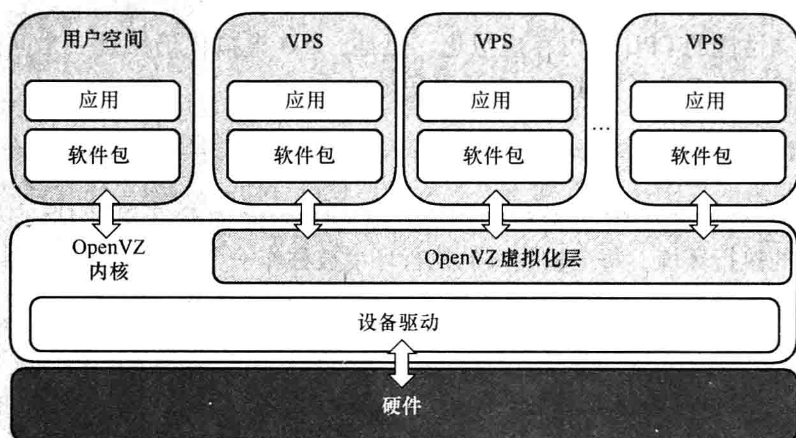


图 1.6 OpenVZ 架构

VPS 的一块虚拟内存。

OpenVZ 虚拟磁盘是主机文件系统的一个分区。类似于 CPU 调度，OpenVZ 磁盘使用是由一个两级磁盘配额确定的。在第一级，OpenVZ 为每个 VPS 定义一个磁盘配额。例如，通过在主机文件系统中限制文件夹的最大尺寸。在第二级，为一个 VPS 中的用户和组定义磁盘配额是可能的，其中使用了标准的 Linux 配额机制。

网络虚拟化层将各 VPS 相互隔离，同时与物理网络隔离 [SWS 05]。OpenVZ 的默认网络虚拟化机制为一个 VPS 创建一个虚拟网络接口，并在主机系统中将一个 IP 地址指派给它。当一条报文（目的 IP 地址为一个 VPS）到达主机系统时，主机系统将该报文路由到相应的 VPS。网络虚拟化的这种方法支持使用主机系统路由模块接收和发送 VPS 报文。这简化了网络虚拟化，但在接下来的路由报文中引入了一个多余的跳。

可做出结论，即 OpenVZ 提供一种高级虚拟抽象，并相比其他虚拟化工具引入较少的额外负担。另外，就必须要共享物理主机系统内核的虚拟环境而言，它的约束更大。

1.3 场景和方法论

本节给出为测试所选用的方法论，并比较虚拟化工具，描述实施过的试验。为提供一个公平的比较，决定运行不同的测试，来评估被虚拟化环境的 CPU、RAM、存储和联网性能。为测量性能，选择实施一项特定任务时每个工具所用的时间。

当实现虚拟化时，从如下事实中出现了几个问题，即硬件是在各 VM 之间分割的，见 1.1 节的讨论。特别与性能测量有关的问题之一是，hypervisor 如何向 VM 提供时间记录（timekeeping）机制 [VMW 08]。在一台个人计算机中跟踪时间有几种方式，如读取基本输入/输出系统（BIOS）时钟、使用 CPU 寄存器 [如时间

戳计数器（TSC）]、从 OS 请求系统时间。对于系统时间记录机制，Xen 不断地将 VM 时钟与 dom0 时钟同步，方法是通过在 domU 和 hypervisor 之间的共享内存发送正确的时间信息。对多数应用而言，那种解决方案是可行的。不过，如果应用具有比 VM 时钟要高的时钟频率，则将存在时间测量误差。出于这个原因，在测试中不考虑来自 VM 的时间记录方法。实施测试的主要思路是，使一台外部计算机负责测量在测试计算机中各 VM 实施一项特定任务所花费的时间。在单 VM 测试中，外部计算机运行一个控制脚本，通过一个安全外壳（SSH）连接向 VM 发出指令，使之启动试验，并数次完成一项特定任务。在发起和完成任务的每个轮次时，VM 使用一条探测报文通知外部计算机。在完成所有轮次之后，外部计算机计算完成特定任务的一个轮次花费时间的均值和方差，这里考虑到 95% 的置信区间。为得到较小方差而选择轮次数，轮次数是任务相关的。在多个 VM 测试中，规程是类似的，区别在于外部计算机同时跟踪多个任务执行的情况。为做到通知外部机器所花费时间量是可忽略的，配置所有测试的任务，使测试的各轮次持续一个大得多的时间量。典型的通知时延在毫秒量级，而轮次执行则在分钟量级。图 1.7 给出单 VM 测试的基本场景，这里考虑仅有一个轮次的一个测试。开动测试（1）指令是从外部计算机发送到测试计算机的。在执行任务之前（3），测试计算机将初始通知（2）发送到外部计算机，并在任务执行完成时，发送一个完成通知（4）。最后，外部计算机处理结果（5）。

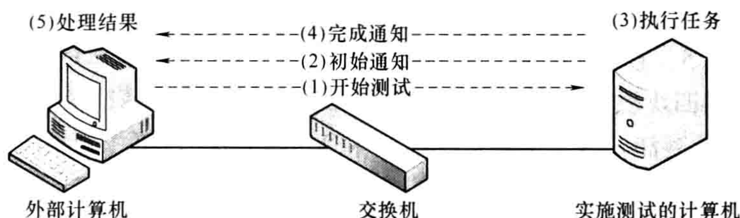


图 1.7 单 VM 测试默认配置

对于网络测试，采用一种不同方法。关注的度量元是当接收和发送时各 VM 取得的吞吐量。对于这些测试，有必要有至少两台计算机，一台用于产生/接收网络流量，另一台实施互补动作（complementary action）。一台额外的计算机被用来实施虚拟化系统的互补动作。这台计算机被称作网络测试计算机。对于这些测试，由网络基准测试工具本身进行测量，所用方案如图 1.8 所示。外部计算机最初触发测试（1）。在接收到指令之后，执行网络基准测试，测试报文要通过网络（2）。在测试完成之后，测试的计算机和网络测试计算机将结果发送到外部计算机（3）处理结果（4）。

下面介绍硬件/软件描述。

对于所实施的测试，见上面的描述，使用三台计算机。这里将描述这些计算机的硬件和软件配置。

测试是在第一台计算机中执行的，该计算机被称作测试的计算机，如图 1.7 和

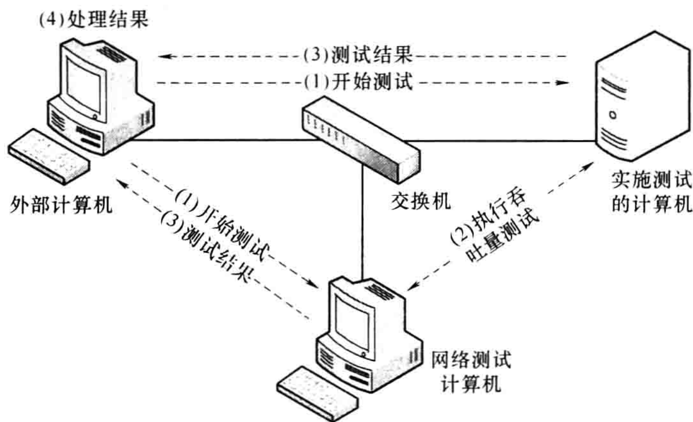


图 1.8　单 VM 网络测试配置

图 1.8 所示。对于那个角色，使用一台 HP Proliant DL380 第 5 代服务器，装备有 2 个 Intel Xeon 5440 CPU、10GB DDR2 RAM @ 667MHz、一个集成的 2 端口 Broadcom Nextreme II 千兆网络接口和 4 个 10000r/min 的 146GB SAS 硬盘。针对测试，每块硬盘安装有一个不同的软件配置。在第一块硬盘中，为原生性能测试的用途，安装一个标准的 Debian Linux AMD 64。在第二块硬盘中，针对 dom0 和 domU，使用一个半虚拟化的 Debian Linux AMD 64，安装一个标准的 Xen 3. 2-1。domU 的各 VM 配置有一个 vCPU、4GB RAM 和 10GB 存储。在第三块硬盘中，安装一个标准的 VMware ESX 3. 5，且 VM 配置有一个 vCPU、4GB RAM、10GB 存储和 Debian Linux AMD 64。在第四块硬盘中，在 Debian AMD 64 之上安装一个标准的 OpenVZ，且创建一个 OpenVZ 容器，其配额类似于 Xen 和 VMware VM。第二台机器是外部计算机，用于控制试验和处理结果。在图中它被称作外部计算机。对于那个角色，使用一台台式机，装备有一个 Intel Q6600 CPU、4GB DDR2 RAM@ 667MHz、一个集成的 Intel e1000e 千兆网络接口和 7200r/min 的一块 500GB SATA2 硬盘。该计算机配置有一个默认的 Debian Linux AMD 64，并有控制任务和处理结果的所有脚本。最后一台计算机是网络测试中使用的计算机，用于建立到测试计算机的一条连接，或接收来自测试计算机的一条连接。在图中被称作网络测试计算机。这台计算机是一台台式机，装备有一个 Intel E6750 CPU、2GB DDR2 RAM@ 667 MHz、一个集成的 Intel e1000 千兆网络接口和运行在 7200r/min 的一块 320GB SATA2 硬盘。该计算机配置有一个默认的 Debian Linux AMD 64 和网络基准测试工具。所有的 Linux 安装使用 2. 6. 26 内核版本。

1.4　性能评估

首先给出采用一个 VM 的试验结果，将采用 Xen、VMware 和 OpenVZ 得到的性

能与原生 Linux 的性能进行比较。针对 CPU、RAM、磁盘和网络资源，使用特定的基准测试。针对一个 Linux 内核编译基准测试，这里也给出结果，这是要求 CPU、RAM 和硬盘等混合资源的一项应用。

1.4.1 CPU 性能

Super-Pi 测试是基于 Gauss-Legendre 算法计算 Pi 值的一项 CPU 密集的任务。Gauss-Legendre 算法是迭代的，且基于多次算术运算。最常用的算术运算是求和、除、方根、乘幂（potentiation）、减和乘。对于这次测试，一个外壳脚本计算具有 2^{22} 个数字（4 194 304 个数字）Pi 值，运行 10 个轮次。性能度量元是计算 Pi 所用时间，单位是 s。

1.4.2 内存性能

内存分配、设置和读取（MASR）是由 Rio de Janeiro 联邦大学电信和自动化组（GTA）开发的一个内存基准测试工具。MASR 基准测试内存的方法是，分配 2GB 内存，顺序地将所有内存位置设置为一个固定值，并顺序地读取所有内存位置。MASR 为基准测试内存开发了确定数量的操作，这独立于计算机的性能。因为对于这个显性特点没有找到 Linux 内存基准测试，所以开发了 MASR。对于这项测试，开发了一个外壳脚本，其中 MASR 被执行 10 个轮次。被评估的参数是在每个轮次中执行 MASR 基准测试所花费的时间。

1.4.3 硬盘和文件系统性能

1. Bonnie ++

Bonnie ++ 是一个开源磁盘基准测试程序，设计用来评估硬盘和文件系统性能。Bonnie ++ 的主程序测试 1GB 大小的单个临时文件，或对于较大量的数据测试多个 1GB 大小的临时文件，带有数据库类型的访问，在所用临时文件中模拟诸如创建、读取和删除大量小文件等操作。第二个程序测试硬盘不同区域的性能，从位于硬盘开始、中间和末尾扇区的块中读取数据。对于这项测试，一个外壳脚本执行 10 次 Bonnie ++，带有使用 2GB 磁盘空间的一个参数。性能度量元是每次执行 Bonnie ++ 基准测试所花费的时间。

2. ZFG

零文件生成器（ZFG）是由 GTA/UFRJ 开发的一个磁盘基准测试工具。通过在每个轮次中 10 次以零填充写入一个 2GB 二进制文件，ZFG 基准测试磁盘连续写入速度。ZFG 是为以确定数量（独立于计算机的性能）的操作基准测试磁盘而开发的。因为没有找到带有这个显性特点的 Linux 磁盘基准测试工具，所以开发了 ZFG。针对这项测试，开发了一个外壳脚本，其中以 10 个轮次执行 ZFG。被评估参数是每个轮次执行 ZFG 基准测试所花费的时间。

1.4.4 网络性能

Iperf是一个开源的网络基准测试程序，支持在传输控制协议（TCP）或用户数据报协议（UDP）之上使用单向和双向数据流来基准测试网络性能。Iperf有可被配置的几个参数，如报文尺寸、带宽（Iperf应该尝试取得的）和测试时长。在测试中使用Iperf，配置有单向UDP数据流，使用1472个字节的净荷来避免IP分片。对于这项测试，开发了一个外壳脚本，其中一条单向UDP数据流从测试计算机传到网络测试计算机或以相反方向传递。

1.4.5 整体性能——Linux内核编译

Linux内核编译［WRI 02］是被频繁地用来评估整体性能的一个基准测试工具，因为它密集地使用计算机系统的不同部分。Linux内核由数千个小型源代码文件组成。其编译要求密集的CPU使用、RAM读/写访问和短时长非顺序的磁盘访问，原因是要被编译的文件取决于目标内核配置。对于这项测试，开发了一个外壳脚本，其中10次编译内核。被评估的参数是产生一个默认配置文件和编译内核所花费的时间。

1.4.6 单个虚拟机测试

首先给出所实施的单VM的执行基准测试的结果。有关单VM测试的所有下面的图形，都将给出原生Linux、VMware VM、Xen VM和OpenVZ容器的性能。

1. Super-Pi

10次执行Super-Pi基准程序的结果如图1.9所示。一个轮次测试的均值执行时间显示在纵轴上，值越小越好。虚拟化工具和原生Linux沿横轴展开。

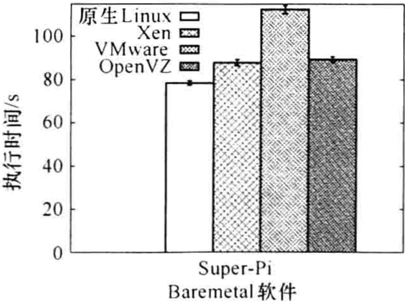


图1.9 Super-Pi测试

和预料的一样，非虚拟化系统的性能胜过虚拟化系统。Xen接近原生性能。那是预料到的情况，原因是在hypervisor赋予使用一个内存区的许可之后，Xen半虚拟化VM可直接访问内存，且将大部分给出的额外负担都与VM CPU调度器关联起来。VMware具有最差的性能，这是提供给VM的虚拟RAM和物理RAM之间额外地址翻译的后果，VM CPU调度器的情况也是这样的。由于在容器之间共享CPU的调度机制，OpenVZ有比Xen稍高一点的额外负担。

2. MASR

MASR基准测试执行10个轮次的结果如图1.10所示。一个轮次的均值执行时

间显示在纵轴上，值越小越好。虚拟化工具和原生 Linux 沿横轴展开。

结果表明，和预料的一样，原生 Linux 具有最佳性能。和预料的一样，OpenVZ 取得类似于原生 Linux 的性能，原因是 OpenVZ 动态地与具有低隔离度的所有容器共享内存，低隔离度仅为每个容器保障最小的私有内存量。Xen 给出与 VM 调度机制相关联的某种额外负担。由于额外内存地址翻译和 VM 调度器，VMware 具有最差的性能。所有虚拟化工具给出可接受的额外负担，这是一个非常重要的结果，原因是在转发表查找和其他路由任务中，内存访问操作是非常常见的。

3. Bonnie ++

执行 10 次 Bonnie ++ 磁盘基准测试的结果如图 1. 11 所示。一个轮次测试的均值执行时间显示在纵轴上，值越小越好。虚拟化工具和原生 Linux 沿横轴展开。

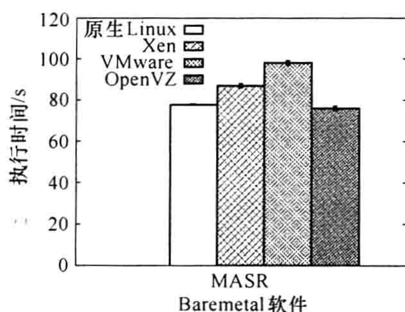


图 1. 10 MASR 测试

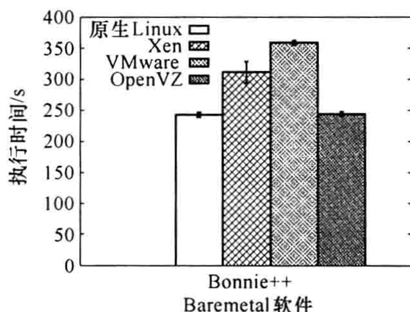


图 1. 11 Bonnie ++ 测试

原生 Linux 具有最佳性能，OpenVZ 取得类似结果，原因是 OpenVZ 中的磁盘访问是非常类似于原生 Linux 中的磁盘访问的。Xen 给出一些额外负担，原因是它使用 dom0 和 VM 之间的一个额外缓冲，即 I/O 环，当从磁盘读取时这增加了延迟。VMware 给出最差的结果，这也许是 hypervisor 中额外缓冲的后果，该缓冲被用来在物理磁盘和虚拟磁盘之间传递数据。不过，磁盘性能是不太重要的，原因在于它多少不会影响正常的路由器转发操作。

4. ZFG

ZFG 测试的结果如图 1. 12 所示。VMware、Linux 和 OpenVZ 的结果是从 10 个轮次中得到的。为得到一个可接受的误差条，从 100 个轮次中得到 Xen 结果。一个轮次的测试的均值执行时间显示在纵轴上，值越小越好。虚拟化工具和原生 Linux 沿横轴展开。

如在图 1. 12 中所观察到的，VMware VM 的性能稍高于原生 Linux，将这一点归于在虚拟磁盘和真实磁盘之间传递数据的缓冲，原因是它赋予 VM 这样的印象，即当实际上将数据从缓冲传递到物理磁盘时，数据已经写入磁盘。令人意外的是，OpenVZ 的性能超过原生 Linux，必须进行更多的深入研究来发现原因。Xen VM 的性能要慢于原生 Linux，但观察到一个不正常的执行时间。在图 1. 13 中，注意到几

个轮次执行花费的时间小于 10s，一些轮次的执行花费的时间大于 60s。在原生 Linux 中，当有数据要被写入到一个 I/O 设备时，它被发送到一个内存缓冲，之后使用直接内存访问（DMA）被传递到该设备。在 Xen 和 VMware 中，当数据实际上处在从 VM 到真实 I/O 设备驱动的缓冲时，用来与真实 I/O 设备驱动的额外步骤（其中虚拟 I/O 设备驱动将数据写到一个内存缓冲）给出数据被写入到硬盘的印象。对于 Xen 实现有关这个缓冲的进一步细节可在文献［CHI 08］中找到。由 Xen 结果给出的巨大轮次执行时间差异验证了假设，因为依据在每个轮次开始中缓存的状态，每个轮次需要的时间将是变化的。为验证怀疑，进行了两种进一步的测试。第一种测试为 VM 缓冲设置不利条件。在这些类型的测试中，修改测试脚本，在每个测试轮次之前，写入大量数据，目的是将从 VM 到真实 I/O 设备驱动的缓冲处于污染状态（dirty）。在这些类型的测试中，在每个轮次的开始处，预期从 VM 到真实 IO 驱动的缓冲被填充以发出的（pending，悬而未决的）请求。第二种测试有利于 VM 缓冲。修改脚本在每个轮次之前等待一个长的时段，赋予 dom0 足够的时间清除缓冲。在这些类型的测试中，预期从 VM 到真实 I/O 驱动的缓冲是空的，即在每个轮次开始处没有发出的请求。

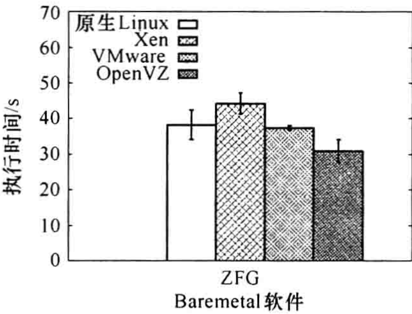


图 1.12　ZFG 测试

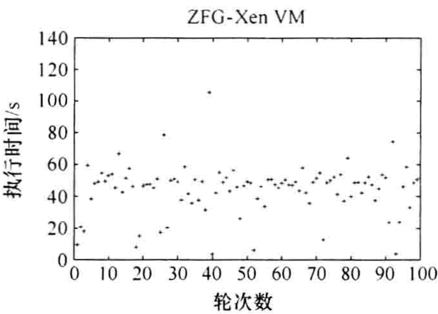


图 1.13　ZFG 测试 Xen VM 各轮次

5. 缓冲污染的 ZFG

在修改过的 ZFG 测试中，在每个轮次之前，以大量数据污染缓冲，其结果如图 1.14 所示。一个轮次测试的均值执行时间显示在纵轴上，值越小越好。虚拟化工具和原生 Linux 沿横轴展开。

进行了 100 个轮次的测试，正如所料的是，当污染在虚拟设备驱动和真实设备驱动之间传递数据的缓冲时，各 VM 的性能急剧下降，原生 Linux 以巨大优势在性能上胜过 Xen 和 VMware 虚拟化 Linux OS。同样，OpenVZ 的性能胜过原生 Linux，表明 OpenVZ 所采用的磁盘机制并不类似于 Xen 和 VMware VM 所采用的磁盘机制。

6. 带有休息时段的 ZFG

在修改过的 ZFG 测试中，在每个轮次之前都等待 dom0 清除缓冲，其结果如图 1.15所示。一个轮次的测试的均值执行时间显示在纵轴上，值越小越好。虚拟

化工具和原生 Linux 沿横轴展开。

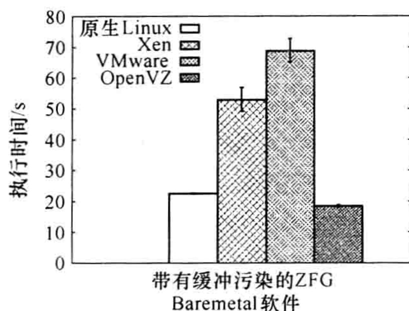


图 1.14 缓冲污染的 ZFG 测试

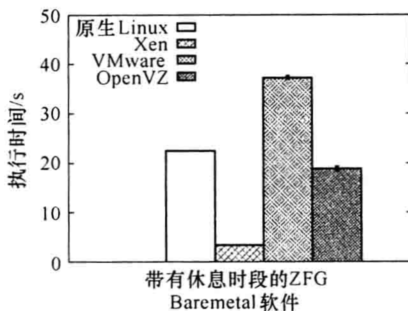


图 1.15 带有休息时段的 ZFG 测试

和预料的一样，当在各轮次之间存在一个长的时段时，Xen 具有足够的时间将处于缓冲中的所有待写出数据写到硬盘，且当执行下一个轮次的测试时，缓冲被清空，给出 Xen VM 将数据较快速地写到磁盘的印象。因此，当要写数据的量不大于缓冲时，Xen VM 具有如此之好的性能，原因是它写到连接虚拟磁盘驱动和真实磁盘驱动的 RAM 内存缓冲上，所以在性能上就胜过实际上将数据写到硬盘的原生 Linux。允许在每个轮次之间有 2min，使 Xen 来清除缓冲。在 VMware 中没有这样的机制，所以其性能也不好。和在前面 ZFG 磁盘写入测试中一样，OpenVZ 再次胜过 Linux。

7. Iperf：从 VM 到网络测试计算机的 UDP 数据流

在这项测试中，对从 VM 产生的 UDP 报文进行基准测试。在下一节给出报文接收的结果。图 1.16 给出 Iperf 网络基准测试 100 个轮次产生从 VM 到网络测试计算机的流量的结果。在纵轴上显示均值吞吐量，值越大越好。虚拟化工具和原生 Linux 沿横轴展开。

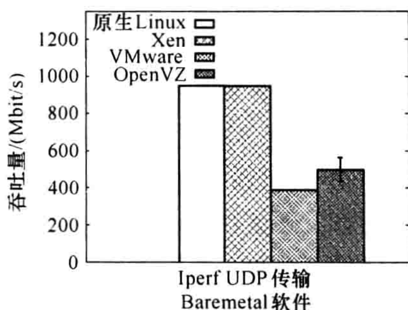


图 1.16 Iperf UDP 报文产生测试

和预期的一样，原生 Linux 取得最佳结果，接近千兆以太网适配器的理论极限。Xen 取得接近原生的性能，表明 I/O 环机制能够足够快地交付数据。VMware 呈现极大的性能降级，表明其默认联网机制的低效实现。基于 IP 层虚拟化的 OpenVZ 默认联网机制，性能较差，且不能使用全部的千兆接口带宽。这是一个非常重要的结果，原因是一台路由器的转发能力直接取决于其发送报文的能力，且结果表明，一个虚拟化环境以接近原生速率发送大型报文是可能的。

8. Iperf：从网络测试计算机到 VM 的 UDP 数据流

在这项测试中，对从 VM 接收 UDP 报文进行基准测试。图 1.17 给出 Iperf 网络

基准测试100个轮次产生从网络测试计算机到VM的流量的结果。在纵轴上显示均值吞吐量，值越大越好。虚拟化工具和原生Linux沿横轴展开。

结果再次表明，原生Linux接近理论最大吞吐量。Xen再次取得接近原生的性能。在传输方面，VMware的性能表现好得多，但仍然要比原生Linux和Xen要差。OpenVZ默认的联网虚拟化机制性能表现较差。这也是一个重要结果，原因是一台路由器的转发能力直接与其接收报文的能力有关。结果表明，Xen能够以接近原生的速率采用单VM处理大型报文接收，在1.4.7节给出多个VM测试。

9. Linux内核编译

图1.18给出了10次执行Linux内核编译测试的结果。一个轮次测试的均值执行时间显示在纵轴上，值越小越好。虚拟化工具和原生Linux沿横轴展开。

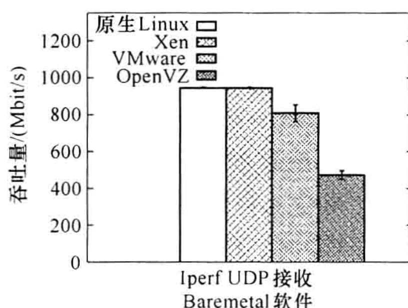


图1.17 Iperf UDP测试

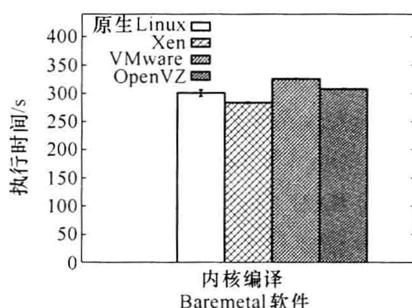


图1.18 内核编译测试

同样，Xen取得的性能稍好于原生Linux的，这源于在前面磁盘写入测试中所解释的磁盘写入效应。对于这项任务，VMware取得最差结果，这重复了在前面测试中差的CPU和内存性能结果。OpenVZ性能类似于原生Linux，这与在前面CPU、内存和Bonnie++磁盘测试中所给出性能预期的一样。

10. 有关单VM结果的评述

要评述的一个重要点是VM的磁盘性能。取决于应用，在真实磁盘驱动和虚拟磁盘驱动间通信的缓冲可以有正面影响或负面影响。对于执行多次磁盘读取操作的应用而言，硬盘和VM磁盘驱动之间的额外阶段，为恢复信息引入较长的时延，因此就降低了性能。如果磁盘写入操作比读取频繁，且应用不写入大量数据以填充缓冲，那么应用应该可从缓冲受益，这正像在各轮次之间有一个时间间隔时所执行的ZFG基准测试的情形。会损失性能的应用包括站点的网站主机（必须频繁地从磁盘恢复信息）或数据挖掘应用（密集地从磁盘读取信息）。会从缓冲受益的应用包括日志应用和主要为写入数据而访问磁盘的其他应用。考虑所给出的结果，特别是CPU、内存访问和网络有关的结果，得出结论，首先，虚拟化工具引入可接受的额外负担，且在被测工具之上构造虚拟路由器是可行的。基于CPU、内存和网络（它们是路由器正常操作所使用的主要资源）结果，得到结论，即VMware不是构

造虚拟路由器的最佳方案，原因是它有最差的性能结果。同样，VMware 几乎不提供灵活性，原因是其源代码是私有的。在多数情形中，OpenVZ 的性能非常好，但呈现出差的网络性能，这是网络虚拟化的一个主要问题。此外，OpenVZ 也约束虚拟环境，限制它不仅使用相同的 OS 而且为所有 VM 都使用相同内核版本，这是以不同路由器复杂度构造虚拟网络的一个主要问题。由于给出良好的 CPU 和内存性能，具有接近原生的网络性能，开源（允许无限制地定制）和以高度灵活性提供虚拟环境，所以 Xen 也许是构造虚拟路由器的最佳虚拟化工具。1.4.7 节给出采用多个 VM 的测试集。多 VM 测试评估 Xen 如何随共享相同计算机资源的多个 VM 扩展，评价多个虚拟网络在一台计算机上的生存能力。

1.4.7 多虚拟机测试

多 VM 测试的目标是观察 Xen 如何随多个 VM 在性能降级、公平资源共享等方面扩展的，以及不同 vCPU 到 CPU 分配方案如何影响总体性能和个体性能。下面所有各图给出 Xen VM 的性能，其中执行不同任务时 VM 数量是变化的，并且对相同任务，CPU 分配方案也是变化的。Xen 支持对各 VM 呈现给虚拟 OS 的 vCPU 如何共享物理 CPU 进行配置。它支持动态 CPU 分配方案和静态分配方案。为研究它们对虚拟化环境扩展性的影响，使用了三种不同的 CPU 分配默认。如前所述，测试计算机装备两个 Xeon E5400 四核 CPU，这将 8 个物理核分配在 hypervisor 和 VM 的需求之间。另一个相关点是，每个 CPU 有其自己的正面（front side）总线，这使每个 CPU 能够独立地访问内存。在每个 CPU 内部，每个核有其自己的 L1 和 L2 缓存，但 L3 缓存是为每对核所共享的。在测试中使用的第一个核分配方案是 Xen 默认方案，其中 hypervisor 动态地将 CPU 物理核分配到 Xen vCPU。在 Xen 默认方案中，可能存在未用的物理核、可能被分配到一个以上 vCPU 的核和恰好分配给一个 vCPU 的核。在表 1.1 中给出了核分配的一个例子。在某个时刻，可能存在未用的物理核、被分配到一个以上 vCPU 的核和分配给单个 vCPU 的核。

表 1.1 使用 Xen 默认方案的物理核分配例子

物 理 核	虚拟机的虚拟 CPU
0	dom0 的 VCPU0 和 VCPU7
1	dom0 的 VCPU4
2	dom0 的 VCPU1
3	VM2 的 VCPU0
4	dom0 的 VCPU5 和 VCPU6；VM4 的 VCPU0
5	dom0 的 VCPU2；VM3 的 VCPU0
6	未用
7	dom0 的 VCPU3；VM1 的 VCPU0

使用的第二种方案是一种静态物理核分配方案。将一个固定的独占核设置给每个 VM vCPU，将一个固定的独占核设置给 4 个 dom0 vCPU，禁止 dom0 的 4 个 vCPU。这种分配的目标是考察当固定物理核（将处理一条 vCPU 请求）时，特别当导致降低数量的 CPU 缓存错误和减少 RAM 内存访问（具有比缓存大得多的延迟）时，是否存在一个相当的性能增益。在表 1.2 中给出了这种方案。

表 1.2　独占核静态分配方案

物　理　核	虚拟机的虚拟 CPU
0	dom0 的 VCPU0
1	dom0 的 VCPU1
2	dom0 的 VCPU2
3	dom0 的 VCPU3
4	VM1 的 VCPU0
5	VM2 的 VCPU0
6	VM3 的 VCPU0
7	VM4 的 VCPU0

使用的第三种方案是一个静态物理核分配，其中将所有 vCPU 设置为共享相同的物理核。这种方案的目标是过载物理 CPU，观察性能降低以及各 VM 之间的资源共享是否为公平的。在表 1.3 中给出了这种方案。每个 VM 和 dom0 使用相同的固定物理核。dom0 仅有一个活跃的 vCPU，所有其他 7 个 vCPU 都被禁止。

表 1.3　共享核静态分配方案

物理核	虚拟机的虚拟 CPU
0	dom0 的 CPU0；VM1 的 VCPU0；VM2 的 VCPU0；VM3 的 VCPU0；VM4 的 VCPU0
1	未用
2	未用
3	未用
4	未用
5	未用
6	未用
7	未用

1. Super-Pi

图 1.19a ~ 图 1.19c 给出了扩展性测试结果。其中使用采用三种不同物理核分配方案的 Super-Pi 基准测试，每种方案执行 10 个轮次。

结果表明，所有三种 CPU 方案给出大得多的变差，因为 VM 数量的增加，表

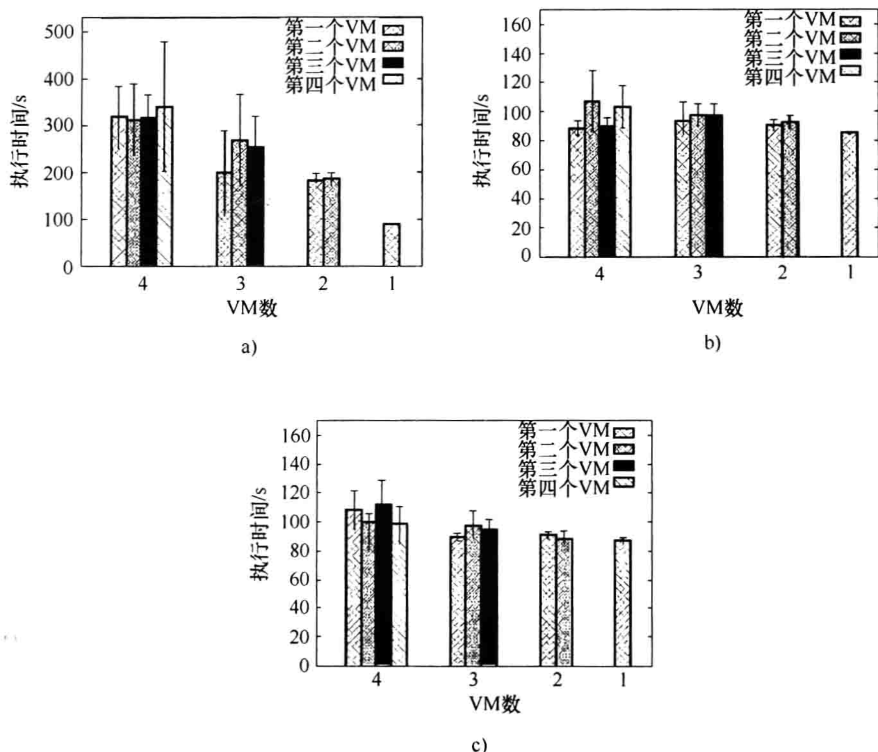


图 1.19 Super-Pi 基准测试

a）对于所有 vCPU 使用一个物理核的多个 VM b）为每个 vCPU 使用一个独占物理核的多个 VM

c）使用 Xen 默认物理核分配方案的多个 VM

明对于这项任务，在以一种公平方式调度各 VM 方面 Xen 存在困难。对于采用一个物理核服务所有 vCPU 的方案，同样显著的是，这也存在最大可观察到的变差，原因是 CPU 是一项稀缺资源，执行的 VM 越多，则一个 Super-Pi 轮次将花费的时间就越长。

2. MASR

图 1.20a ~ 图 1.20c 给出了使用 MASR 基准测试时扩展性测试的结果。其中采用三种不同物理核分配方案，每个执行 10 个轮次。

结果表明，Xen 方案与为每个 VM 和每个 dom0 vCPU 都采用一个独占物理核的方案，可良好地从一个 VM 扩展到 4 个 VM，表明随着 VM 数量增加，没有性能降级。当从一个 VM 扩展到 4 个 VM 时，Xen 默认方案没有损失多少性能，但相比为每个 VM 和每个 dom0 vCPU 都采用一个独占物理核的方案，则给出差的性能。那是如下事实的一个后果，即 Xen 默认方案动态地将 vCPU 重新指派到物理核上，当一个 vCPU 重新指派到一个不同的核时，这增加了缓存不命中数。为所有 VM 和 dom0

使用一个物理核的方案，表明对于这项测试，CPU 是一个瓶颈，且就各 VM 间分割 CPU 而言，Xen hypervisor 是公平的。

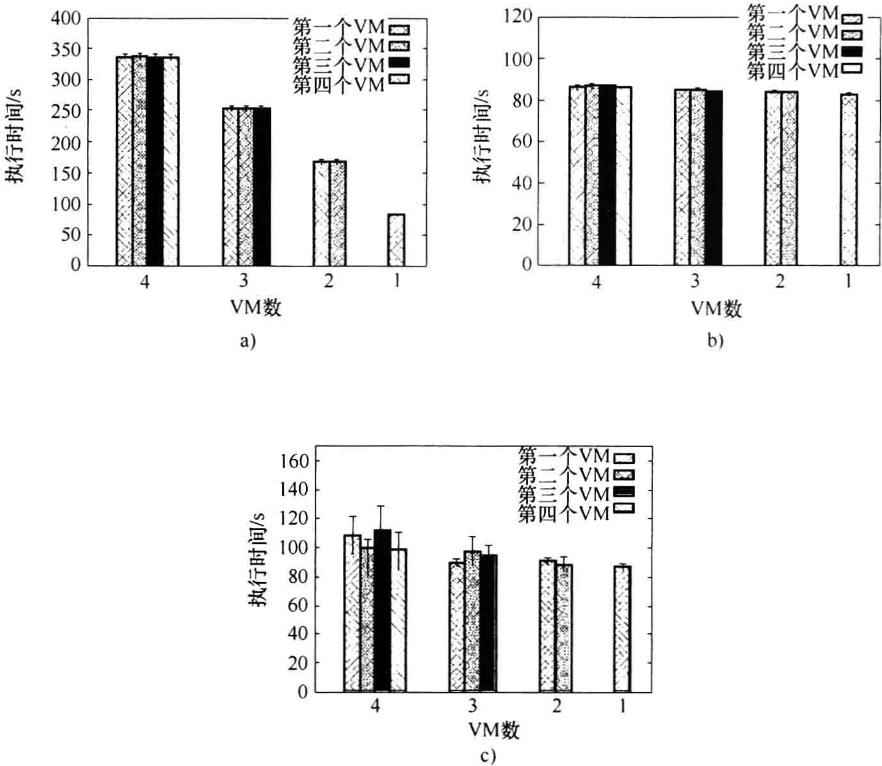


图 1.20 MASR 基准测试

a）对于所有 vCPU 使用一个物理核的多个 VM b）为每个 VM vCPU 和每个 dom0 vCPU 使用一个独占物理核的多个 VM c）使用 Xen 默认物理核分配方案的多个 VM

3. 采用缓冲污染的 ZFG

单 VM 测试揭示出，各 Xen VM 实际上没有同步地将数据写到硬盘。因此，选择仅执行磁盘测试，这是尽可能接近同步写入硬盘的情况。在这项测试中，在每个轮次开始，将从 VM 到真实硬盘驱动的缓冲污染（写入新数据）。图 1.21a ~ 图 1.21c给出了使用缓冲污染 ZFG 基准测试的扩展性结果。其中采用三种不同的物理核分配方案，每个执行 10 个轮次。

结果表明，随着 VM 数增加，所有三种 CPU 方案都给出较大的变差，这表示对于这项任务，在公平地调度各 VM 方面，Xen 是有困难的。给出最佳性能的方案是针对每个 vCPU 采用一个独占物理核的方案。当相比于针对每个 vCPU 有一个独占的物理核时，以一个物理核服务所有 vCPU 的方案给出较小的额外负担。这可由如下事实解释，即在这项任务中，瓶颈是硬盘，即使对所有 vCPU 仅有一个核时也是如此。令人惊奇的是，当该任务采用四个 VM 执行时，Xen 默认方案给出低的性

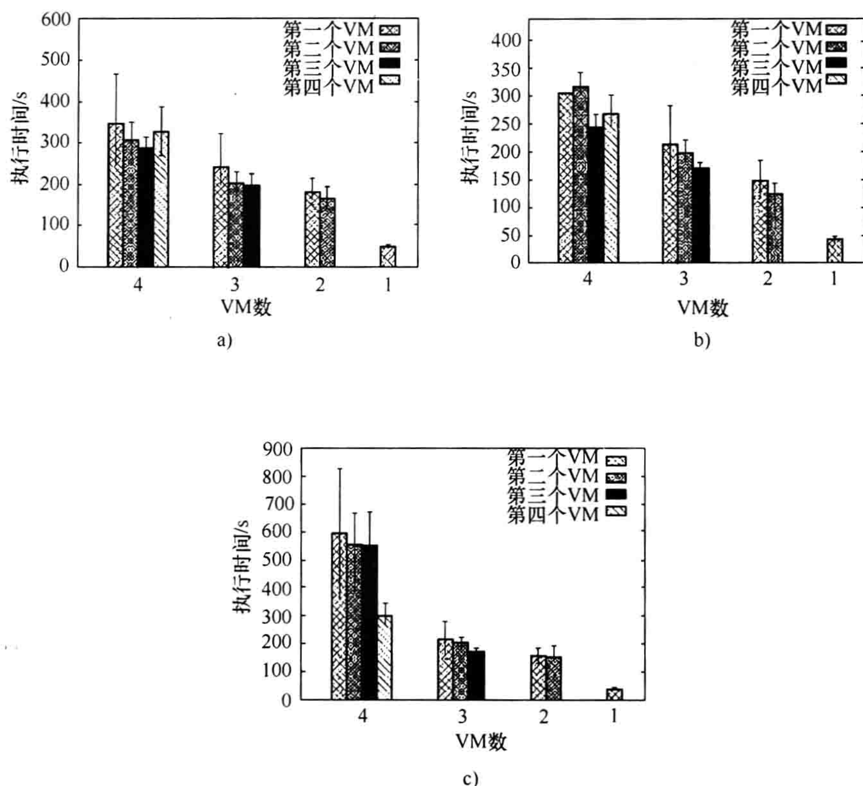


图 1.21 采用缓冲污染的 ZFG 基准测试

a）对于所有 vCPU 使用一个物理核的多个 VM b）为每个 VM vCPU 和每个 dom0 vCPU 使用一个独占物理核的多个 VM c）使用 Xen 默认物理核分配方案的多个 VM

能，这显示了不稳定性。为理解为什么观察到这种不稳定性，需要进行进一步的深入研究。

4. Iperf：从 VM 到网络测试计算机的 UDP 数据流

图 1.22a ~ 图 1.22c 给出了使用 Iperf 基准测试程序产生从 VM 到网络测试计算机的 UDP 数据流时的扩展性测试结果。其中采用三种不同的物理核分配方案，每种方案运行 10 个轮次。

结果表明，Xen 默认物理核分配方案和每个 VM vCPU 一个独占物理核的方案，具有类似的性能，取得的总吞吐量接近网络适配器的理论极限。对于所有 vCPU 一个物理核的方案，瓶颈不是网络适配器而是 CPU，原因是各 VM 不能产生足够的报文来填满网络适配器容量。在所有方案中，Xen 证明了在资源共享方面是公平的。

5. Linux 内核编译

图 1.23a ~ 图 1.23c 中给出了使用 Linux 内核编译的扩展性测试结果。其中采

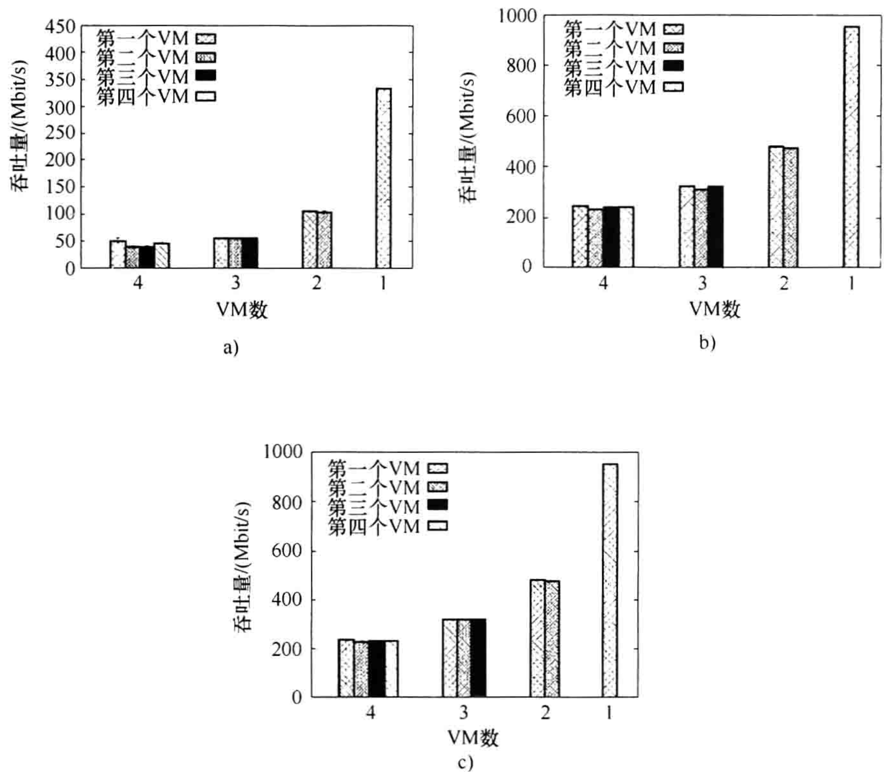


图 1.22 Iperf：从 VM 到网络测试计算机的 UDP 数据流

a）对于所有 vCPU 使用一个物理核的多个 VM b）为每个 VM vCPU 和每个 dom0 vCPU 使用一个独占物理核的多个 VM c）使用 Xen 默认物理核分配方案的多个 VM

用三种不同物理核分配方案，每个执行 10 个轮次。

结果表明，Xen 默认方案以及为每个 VM 和每个 dom0 vCPU 使用一个独占物理核的方案，均可良好地从一个 VM 扩展到四个 VM，显示出没有性能降级。为所有 VM 和 dom0 使用一个物理核的方案表明，对于这项测试，CPU 是瓶颈，且就将 CPU 在各 VM 间进行分割而言，Xen hypervisor 是公平的。

6. 结论

一般而言，当扩展到多个 VM 时，Xen 被证明是公平的。对于一些测试，扩展多个 VM 影响到公平性，这由随着 VM 数增加，结果中增加的方差得到证明。同样被注意到的是，除了对于硬盘测试，将所有 vCPU 集中到一个物理核会相当地降低 VM 的性能。除了内存测试外，在 Xen 默认方案和每个 VM vCPU 一个物理核的方案的性能之间没有显著差异，这表明对于执行的任务，CPU 缓存预留不是一个主要问题。需要额外深入研究的一项意外结果是，当由四个 VM 执行测试时，硬盘测试中 Xen 默认方案的性能较差。

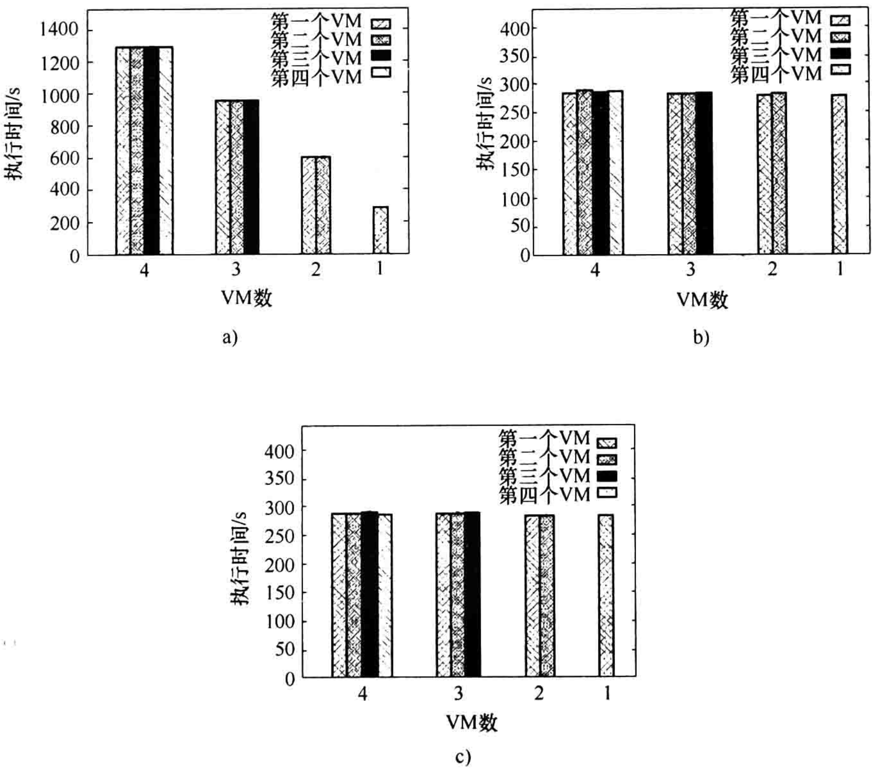


图 1.23 Linux 内核编译

a）对于所有 vCPU 使用一个物理核的多个 VM b）为每个 VM vCPU 和每个 dom0 vCPU 使用一个独占物理核的多个 VM c）使用 Xen 默认物理核分配方案的多个 VM

1.5 小结

建议新互联网架构的研究工作被分成两类：一元论或纯粹论法（其中所提供架构由单个通用网络组成，它支持任何服务需求）和多元论法（其中所提供架构由多个网络组成，构建每个网络来处理一种服务需求）。在本章中，分析了在支持多元论架构中当前硬件虚拟化技术的角色作用。

作为实现路由器虚拟化的候选者，比较了 VMware、Xen 和 OpenVZ。得出结论，VMware 不是路由器虚拟化的最佳方案，原因是在测试中它得到最差结果，而且它是不允许修改的一种私有解决方案（因为不能得到它的源代码）。也得出结论，虽然在多数测试中 OpenVZ 具有极好的性能结果，因为它取得低的网络性能，所以也不是最佳选择，网络性能是构造虚拟路由器的主要考虑之一。OpenVZ 也为虚拟路由器的构造提供低的灵活性，这是因为所有虚拟环境都必须不仅共享相同的

OS 而且要有相同的内核。Xen 为虚拟路由器需求提供最佳匹配（best fit）。因为 Xen 是一个开源工具，所以它为在各 VM 内支持任意 OS 和支持定制方面提供灵活性。Xen 也显示出良好的 CPU、内存和虚拟化网络性能，这些是虚拟路由器转发操作的最重要资源。此外，Xen 在各 VM 间提供公平的资源共享，并可从一个 VM 扩展到四个 VM，没有由 hypervisor 导致的性能降级。

1.6 参考文献

[ADA 06] ADAMS K., AGESEN O., "A comparison of software and hardware techniques for x86 virtualization", *ASPLOS-XII: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[BAR 03] BARHAM P., DRAGOVIC B., FRASER K., *et al.*, "Xen and the art of virtualization", *Proceedings of the 19th ACM Symposium on Operating Systems Principles – SOSP03*, October 2003.

[CHI 08] CHISNALL D., *The Definitive Guide to the Xen Hypervisor*, Prentice Hall, 2008.

[EGI 07] EGI N., GREENHALGH A., HANDLEY M., *et al.*, "Evaluating Xen for router virtualization", *International Conference on Computer Communications and Networks – ICCCN*, pp. 1256–1261, August 2007.

[KOL 06] KOLYSHKIN K., *Virtualization in Linux*, 2006.

[MEN 06] MENON A., COX A.L., ZWAENEPOEL W., "Optimizing network virtualization in Xen", *USENIX Annual Technical Conference*, pp. 15–28, May 2006.

[POP 74] POPEK G.J., GOLDBERG R.P., "Formal requirements for virtualizable third generation architectures", *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[SWS 05] SWsoft Inc., *OpenVZ Users Guide*, 2005.

[VMW 05] VMWare Inc., *VMware ESX Server 2 Architecture and Performance Implications*, 2005.

[VMW 07a] VMWare Inc., *The Architecture of VMware ESX Server* 3i, 2007.

[VMW 07b] VMWare Inc., *A Performance Comparison of Hypervisors*, 2007.

[VMW 08] VMWare Inc., *Timekeeping in VMware Virtual Machines*, 2008.

[WRI 02] WRIGHT C., COWAN C., MORRIS J., *et al.*, "Linux security modules: general security support for the linux kernel," *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[XEN 07] XenSource, Inc., *A Performance Comparison of Commercial Hypervisors*, 2007.

第 2 章　虚拟网络接口

未来互联网架构提案可分为两种主要方法：一元论和多元论。在一元论模型中，如图 2. 1a 所示，网络有一个整体架构，它必须足够灵活，以便支持所用种类的将出现的应用。在这种方法中，每次仅有一个协议栈运行在物理基层之上。另外，多元论方法，如图 2. 1b ［AND 05］所示，是基于互联网必须同时支持多个网络的思想的，每个网络运行适合一项给定应用之需要的一个协议栈。创建针对特定服务而设计的特定网络的做法，简化了要求诸如安全、移动性或服务质量等特征的新应用的部署。我们声称，为提供不同服务而设计多个网络的做法，比为同时处理所有不同服务而设计一个独特网络，要容易。因此，多元论方法可被理解为基于“分而治之”方法的一把伞（伞形结构）。如果寻找可涵盖所有可能需求的单一解决方案是非常困难的，那么可构造带有支持这些需求的多个网络就是一种可行的替代方法。这个特点是多元论架构的主要论点，原因是一元论方法不仅必须解决所有的已知互联网问题，而且它也要演进成在未来是可持续运行的。此外，多元论模型的另一项关键优势是其固有的后向兼容性，原因是当前互联网可以是并行运行的被支持网络之一。

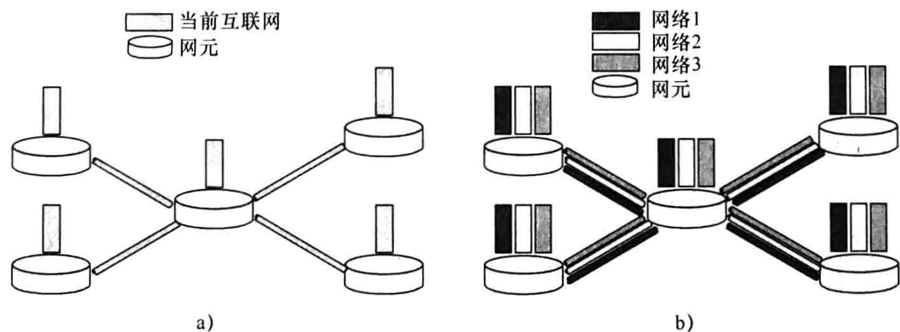


图 2. 1　网络架构：一元论模型（仅支持一个协议栈）和多元论模型（支持数个协议栈）
a）一元论模型　b）多元论模型

多元论提案基于多个网络运行在同一基础设施之上的同一思想［MAC 11，FEA 07，VER 07］，即使它们在报文格式、寻址方案和协议方面存在差异也是如此。因为这些网络共享同一个物理网络基层（如路由器和链路），所以对基础介质的多种访问必须由一个监测器进行编排（orchestrated）。这个监测器是一个特殊用途的软件，它将共享介质虚拟化提供给运行于其上的多个网络。因此，每个网络的运行，就像它正在使用给定量的物理资源一样。这些网络被称作虚拟网络，作为使用一个共享物理基层的结果，它们必须处理性能挑战。相比于一元论架构，这项附

加任务就是多元论架构的额外负担，原因是它添加了一个软件层。

基于多元论方法，以前的工作（如 Horizon 项目团队开发的工作 [HOR 10]）建议使用两个不同的虚拟化平台，提供虚拟联网：Xen 和 OpenFlow。这些平台以一种不同方式攻克在多个虚拟网络间物理资源共享的问题。这些差异导致性能和灵活性之间的折中，这将进一步讨论。

在本章，2.1 节首先给出两个虚拟化平台：Xen 和 OpenFlow。接着，给出焦点在每个平台主要特点上的性能分析。之后，描述系统管理的接口，它们必须足够友好（以便支持多个虚拟网络间的简单资源共享管理），同时也必须方便其上的动作（操作）（如虚拟网络拓扑重新配置）。Horizon 项目为 Xen 和 OpenFlow 平台设计和开发了接口，分别是 VNext [PIS 11] 和 OMNI [MAT 11]，并出于概念验证的目的，为每个平台建立了一个原型 [HOR 10]。所开发的多数接口不需要人类干预，也可由计算机代理使用，自治地控制网络。结果是，这些接口可被用于高级的引导系统，它们能够提供自治的联网服务。2.2 节详细描述 Xen 原型中使用的接口，而 2.3 节描述 OpenFlow 原型的接口。最后，2.4 节给出本章的结论。

2.1 虚拟网络：隔离、性能和趋势

本节描述不同虚拟网络间共享网络物理基层的问题[⊖]。给出虚拟化物理网络的两种代表性方法（Xen [EGI 07] 和 OpenFlow [MCK 08]）的分析，并讨论针对虚拟联网使用这些技术。主要目标是深入考察网络虚拟化的优点和缺点，其中网络虚拟化用作未来互联网的多元论架构的基础。到此为止，实施了各项试验，评估作为一个虚拟化软件路由器时 Xen 和 OpenFlow [MAT 09] 的性能。依据给出的结果和以前的工作，发现了灵活性和性能之间的折中，表明当使用虚拟网络时，共享数据平面的使用可以是一项重要的架构选择。另一个关键点是，使用共享的数据平面，在没有任何可测量到的性能损失情况下，相比于单个虚拟网元处理相同报文速率的场景，Xen 和 OpenFlow 可复用数个虚拟网络。

2.1.1 网络虚拟化方法

将虚拟化看作一个资源抽象，这支持将一项物理资源分片成具有不同能力的相同类型的几项虚拟资源，如图 2.2 所示。这个抽象经常由一个软件层实现，该层提供非常类似于真实接口的“虚拟分片式接口”。在相同资源之上几个虚拟分片的共存是可能的，原因是虚拟化层将真实资源和上面的层进行了解耦。图 2.2 给出了虚拟化的两个例子：计算机虚拟化和网络虚拟化。计算机虚拟化抽象是由虚拟机监测

⊖ 本节部分内容在以前的工作中出现过 [FER 11]。

器（VMM）实现的，它以非常类似于一个计算机硬件接口的一个接口（硬件抽象层）提供虚拟机（VM）。这个接口包括一个处理器、内存、输入/输出（I/O）设备等。因此，每个VM具有直接运行在物理硬件之上的印象，但实际上物理硬件是在几个VM间共享的。我们称这种类型的资源共享为分片，原因在于各VM是隔离的：一个VM不能干扰另一个VM。计算机虚拟化被广泛用于数据中心，其中在单台物理机器中运行几台服务器（软件）。除了节省能量和降低维护成本外，这项技术具有（作为其最重要的特征）灵活性，使每台VM都有其自己的操作系统、应用、配置规则和管理规程。在每个虚拟分片内运行一个定制的协议栈的可能性，是将虚拟化应用到网络的主要动机［AND 05］。如图2.2所示，网络虚拟化类似于计算机虚拟化，其中共享的资源是网络而已。这个概念不是新概念，它已经用在虚拟专网（VPN）和虚拟局域网（VLAN）之中。但是，如今甚至存在新的技术支持路由器虚拟化。在这种情形中，每个虚拟路由器分片可实现一个定制的网络协议栈。

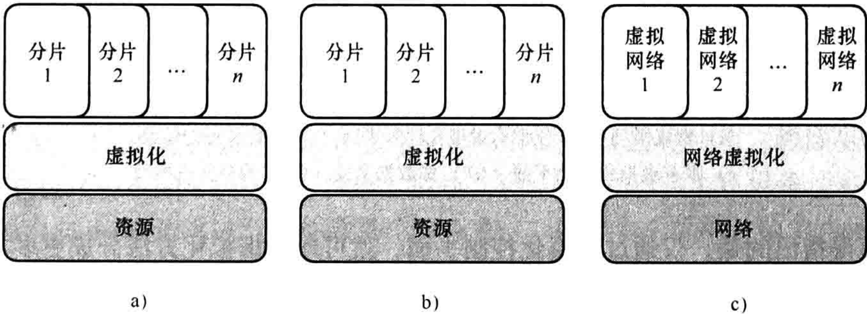


图2.2　虚拟化："分片的"资源、虚拟机和虚拟路由器
a）通用虚拟化　b）计算机虚拟化　c）网络虚拟化

网络虚拟化的方法在它们提供的抽象级别上存在差异，这可由在架构设计中虚拟化层所处位置加以刻画。图2.3比较了对一个网元实施虚拟化的两种基本方法。图2.3a给出常规网元架构，有单一控制平面和数据平面。在一台路由器中，控制平面负责运行网络控制软件，如路由协议［如路由信息协议（RIP）、开放最短路径优先（OSPF）和边界网关协议（BGP）］和网络控制协议［如互联网控制消息协议（ICMP）］，而数据平面负责实现转发表和硬件路径。对路由规程实施虚拟化，意味着在网元架构的某个层次插入虚拟化层，在单一物理网元之上支持多个虚拟网元的共存。假设在控制平面和数据平面之间存在一个虚拟化层，那么仅有控制平面被虚拟化，如图2.3b所示。在这种情形中，数据平面由所有虚拟网络共享，而每个虚拟网络运行其自己的控制软件。相比于常规网络架构，这种方法极大地改进了网络可编程能力，原因是它支持运行多个定制的协议栈，而不是单个默认的协议栈。例如，对网络1、网络2和网络3编程不同网络协议是可能的，如图2.3b所示。在第二种网络虚拟化方法中，控制平面和数据平面都被虚拟化，如图2.3c所

示。在这种情形中，除了控制平面外，每个虚拟网元都实现其自己的数据平面，这甚至更加改进了网络可编程能力。这种方法以报文转发性能为代价，支持数据平面定制，原因是数据平面不再专用于一项共同的任务。在 2.1.3 节的前两部分详细讨论了网络可编程能力和性能之间的折中。

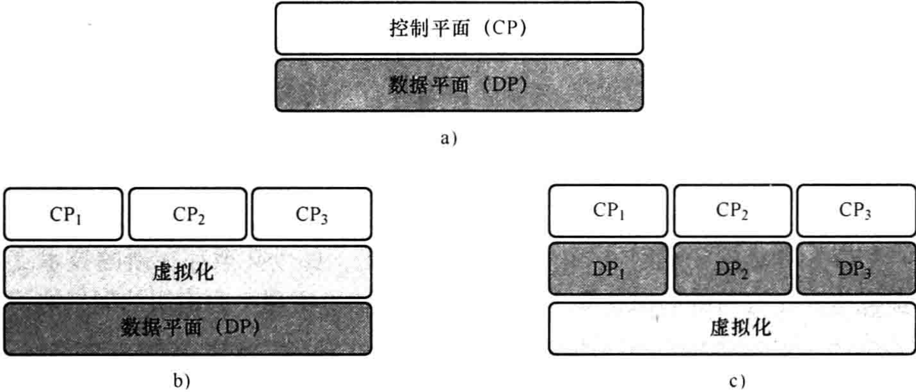


图 2.3 网络模型

a）常规模型 b）仅带有虚拟化控制平面（CP）的多元论模型

c）带有虚拟化控制平面（CP）和数据平面（DP）的多元论模型

值得指出的是，仅通过虚拟化控制平面，就可能将虚拟化方法分成更多类，这取决于在虚拟网元间共享之数据平面的隔离等级。如果要求一种强隔离，那么每个虚拟控制平面必须仅可访问它的数据平面分片，不能干扰其他分片。另外，如果整个数据平面是在虚拟控制平面间共享的，那么就可能出现一个虚拟控制平面干扰其他虚拟控制平面的情况。例如，单个虚拟控制平面可以其自己的表项填充整个转发表，这会导致其他虚拟网络丢弃它们的报文。

2.1.2 网络虚拟化技术

在本节详细给出可用于网络虚拟化的两项技术：Xen 和 OpenFlow。

Xen 是一个开源 VMM，也称作 hypervisor，它运行在商用硬件平台上［EGI 07］。Xen 架构由一个 VMM（位于物理硬件之上）和几个域（同时运行在 hypervisor 之上，也称作 VM）组成，如图 2.4 所示。每个 VM 可有其自己的操作系统和应用。VMM 控制多个域到硬件的访问，也管理这些域间的资源共享。因此，VMM 主要任务之一是隔离不同 VM，即一个 VM 的执行一定不要影响其他 VM 的性能。另外，为提供可靠的和高效的硬件支持，所有设备驱动都保持在称为域 0（dom0）一个孤立的驱动域中［EGI 07］。相比其他域［称作非特权域（domUs）］，域 0 具有特殊权限，原因是它具有物理机器硬件的全部访问权限。非特权域，也称作用户域，拥有虚拟驱动，其运行方式就像它们可直接访问一样。不过，这些虚拟驱动与

dom0 通信，dom0 具有物理硬件的访问权。

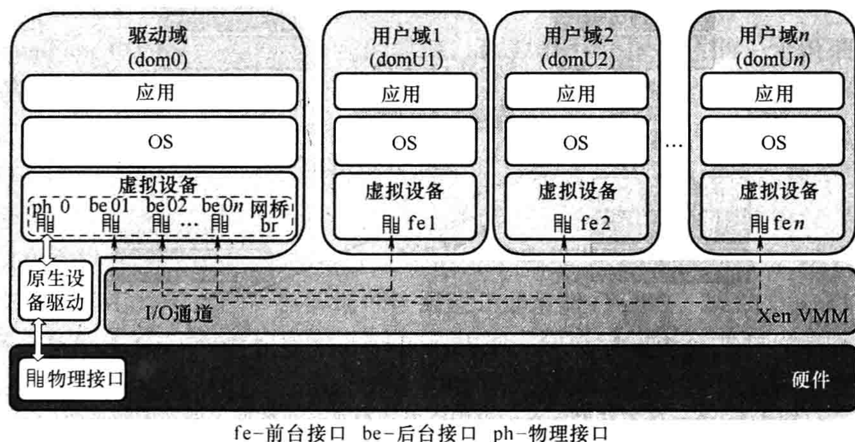


图 2.4 Xen 架构：虚拟机监测器（VMM）、驱动域（dom0）和用户域（domUs）

Xen 虚拟化单个物理网络接口，方法是将来自物理接口的到达报文解复用到 domUs，并在相反方向，复用由这些 domUs 产生的外发报文。这个规程，称作网络 I/O 虚拟化，工作过程如下。域 0 直接访问 I/O 设备（通过使用它的原生设备驱动），它也代表 domUs 执行 I/O 操作。另外，domUs 利用虚拟 I/O 设备（由虚拟驱动控制）向 dom0 请求进行设备访问［MEN 06］，如图 2.4 所示。每个 domU 有其自己的虚拟网络接口，称作前台接口，是网络通信所需要的。后台接口是在 dom0 中对应于一个 domU 的每个前台接口创建的，并在 dom0 中作为虚拟接口的一个代理。前台接口和后台接口是通过一个 I/O 通道相互连接的，该通道采用一种零复制机制将包含报文的物理页重新映射到目标域。采用这种方式，在后台接口和前台接口之间交换报文［MEN 06］。前台接口可由运行在 domUs 中的操作系统检测到作为真实的接口。不过，dom0 中的后台接口是连接到物理接口的，且通过一个虚拟网桥相互连接。这是 Xen 使用的默认架构，称作网桥模式。因此 I/O 通道和网桥在 domUs 中创建的虚拟接口和物理接口之间建立一条通信路径。

不同虚拟网元可使用 Xen 加以实现，原因是它支持多个 VM 同时运行在相同硬件上［EGI 07］，如图 2.5 所示。在这种情形中，每个 VM 运行一台虚拟路由器，它有其自己的控制平面和数据平面，原因是 Xen 虚拟化层处在一个较低层次。

OpenFlow［MCK 08］在大学校园上支持使用布线机柜，这不仅可用于生产网络，而且可用于试验网络。OpenFlow 项目是由斯坦福大学提出的，目标是与生产网络并行地为创新建立虚拟环境，其中使用诸如交换机、路由器、接入点和个人计算机等网元。

OpenFlow 为提供虚拟网络环境给出一种新架构。主要思想是控制平面和数据

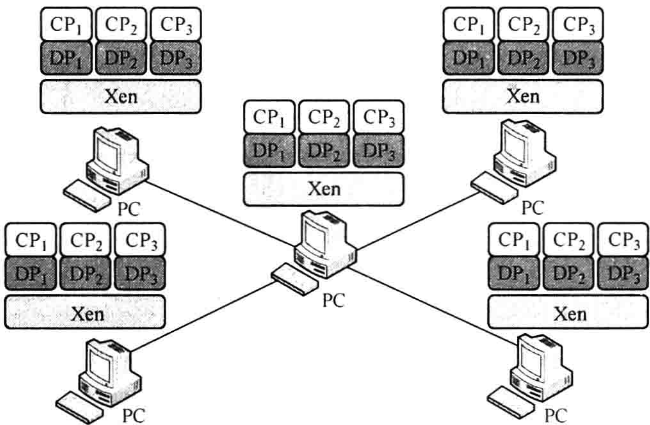


图 2.5 Xen 虚拟联网：每台虚拟路由器都有一个数据平面和控制平面

平面的物理隔离。因此，不同网元执行报文转发规程（数据平面）和网络控制规程（控制平面）。转发单元的虚拟化是由一个共享的流表完成的，它代表数据平面，而所有控制平面是以中心方式处在称作控制器的一个节点中的，它运行控制每个虚拟网络的应用。OpenFlow 运行的一个例子如图 2.6 所示。

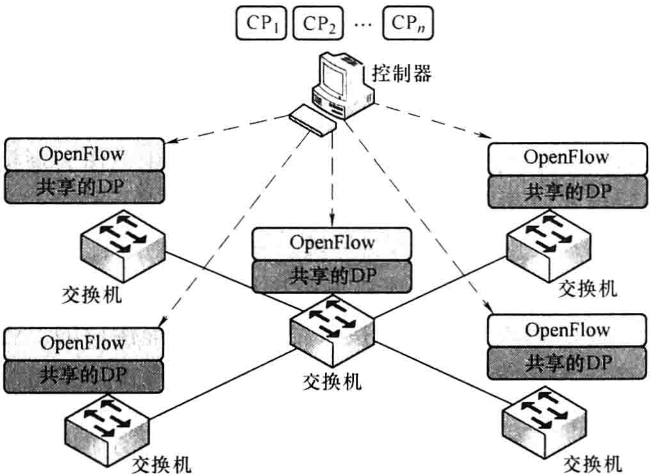


图 2.6 OpenFlow 虚拟联网：每个节点一个共享的数据平面，所有的控制平面都在一个节点中

OpenFlow 协议定义转发节点和网络控制器之间的通信。OpenFlow 基于每个转发节点和控制器之间一条安全信道的建立上，控制器使用这条信道监测和配置转发节点。每次有一条新的报文到达一个转发单元且没有以前配置的流时，该报文的前面一些位被转发到控制器，控制器在所选中的转发单元中为这条报文设置一条路径。控制器也依据常规层 2（L2）和层 3（L3）路由为要转发的一条流设置正常的

处理动作，就像不存在 OpenFlow 一样。这就是为什么在不影响正在进行的流量条件下，OpenFlow 可与生产网络并行使用的原因。

OpenFlow 中的数据平面是由首部字段、计数器和动作描述的一个流表。首部字段是描述报文首部的一个 12 元组的结构，如图 2.7 所示。通过为每个字段设置一个值或使用一个通配符仅设置各字段的一个子集，这些字段可指定一条流。流表也支持子网掩码的使用，如果所用硬件也支持这种匹配 [PFA 09]。这个 12 元组结构为报文转发赋予高度灵活性，因为一条流不仅基于目的互联网协议（IP）地址 [像在常规传输控制协议（TCP）/IP 网络中那样]，而且基于 TCP 端口、介质访问控制（MAC）等。因为各条流可依据层 2 地址进行设置，所以 OpenFlow 的转发单元也被称作 OpenFlow 交换机。但是，这不一定就意味着报文转发是在层 2 进行的。OpenFlow 未来目标之一是处理用户描述的首部字段，这意味着报文首部将不仅由固定字段所描述，而且由虚拟网络管理员所指定的字段组合所描述。这将赋予 OpenFlow 如下能力，即对属于运行任意协议栈的网络的报文进行转发。



图 2.7　一个 OpenFlow 网络中的一条流表项

在首部字段之后，流描述后跟有计数器，它们用于节点监测。计数器计算诸如流时长和所转发字节总数等数据。流描述中的最后字段是一些动作，是在转发单元中一条特定流的每条报文上可采取的指令集合。这些动作不仅包括将一条到达的报文交换到一个目的端口，而且包括改变首部字段（诸如 VLAN 标识符以及源和目的地址）。

控制器节点是网络中的一个中央单元，可与所有节点通信，配置它们的流表。控制器运行一个网络操作系统，它提供虚拟网络管理应用，这是网络配置的基本功能。因此，OpenFlow 中的控制器是作为网络应用和转发单元之间的一个接口发挥作用的，提供访问来自一条流的各条报文和监测节点的基本功能。OpenFlow 可与兼容于 OpenFlow 协议的任何控制器一起工作，如网络操作系统（NOX） [GUD 08]。在这种情形中，每个控制平面是由运行于 NOX 之上的一个应用集合组成的。因此，OpenFlow 中的一个虚拟网络是由其控制平面定义的，是运行于控制器之上的一个应用集合，且其相应流处于控制之下，如图 2.8 所示。

使用单控制器模型，就可能创建许多虚拟网络。但是，注意，运行于同一操作系统之上的不同应用是没有隔离的。结果是，如果一个应用存在某个缺陷，则它可终止控制器，损害所有其他虚拟网络。FlowVisor 是与 OpenFlow 一起使用的一个工具，支持不同控制器作用在同一物理网络之上 [SHE 10]。FlowVisor 作为转发单元和各控制器之间的一个代理，其中假定（例如）每个网络一个控制器。使用这个

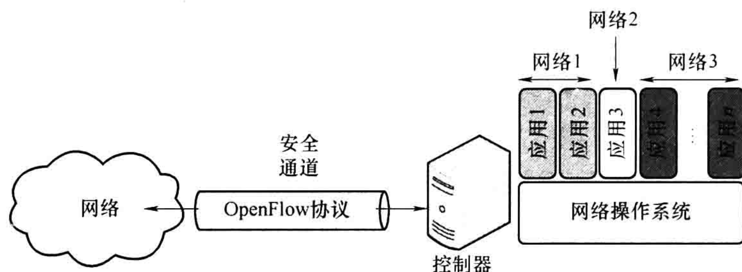


图 2.8 中心式 OpenFlow 控制器模型

模型，确保一个虚拟网络中的故障不会影响其他虚拟网络，就是可能的。

基于分布式转发单元（已经为运行一个网络提供基本功能）和中心式控制平面的思想，OpenFlow 提供一个灵活的基础设施。使用这个基础设施，将物理网络分片成多个虚拟网络就是可能的。在 OpenFlow 中，一个网络的实例化仅是在控制器中创建应用的某个集合。依据进入网络的报文，将应需地创建新的网络。OpenFlow 也为重新分配网络资源提供一个灵活的基础设施。在 OpenFlow 中重新分配一个网络仅意味着对每个参与节点的流表重新编程。对于控制器，这是一项简单的操作，原因是它知道物理设备处在哪里以及它们是如何连接的。

2.1.3 Xen 和 OpenFlow 网络虚拟化技术的特征

Xen 和 OpenFlow 都不是针对支持互联网多元论架构而开发的。不过，迄今为止，它们是一个虚拟网络基层的最佳商用方案。评估这些技术的主要特征，突出它们在支持多个网络和为创新提供灵活性方面的优势和劣势。

Xen 和 OpenFlow 具有不同的虚拟化概念。Xen 通过将物理网络单元分片成不同的并发虚拟路由器而创建虚拟网络。结果是，将一个虚拟网络看作分布于物理基础设施之上互联的虚拟路由器集合。另外，OpenFlow 通过将网络控制分片成多个控制平面（设置每台交换机的转发表）而创建虚拟网络。因此，使用 OpenFlow，一个虚拟网络被定义为具有共性特征的流集合，它们由运行在 OpenFlow 控制器上的相同应用集合编排使用。Xen 和 OpenFlow 虚拟化模型之间的差异，影响扩展性、报文处理、报文转发和基本管理工具的使用。

1. 可编程能力和网络处理

多元论方法的主要优势之一是支持创新。作为一个结果，网络必须足够灵活，以便在可用物理基础设施之上提供端到端路径，并确保管理员对网络的全部控制权限，这包括协议栈的选择、转发规则和报文处理等。

因为 Xen 虚拟化层是直接运行在硬件之上的，所以每个虚拟路由器都可访问所有计算机部件，如内存、处理器和 I/O 设备。因此，网络管理员可自由地通过虚拟化层选择每种部件。因此，可为每个虚拟网络定义不同的操作系统、转发表、转

发规则等。此外，数据平面和控制平面可是完全虚拟化的，这样得到网络控制和管理的一个功能强大的和灵活的平台，如图2.3c所示。通过使用Xen，就可能添加新的功能，如逐跳报文处理，这是针对互联网中认证和访问控制的一个重要特征。添加依据报文的签名，将解决原互联网的遗留安全问题，在原互联网中因为网络的“僵化”，这个问题是不能有所改变的［CLA 04］。

OpenFlow虚拟化模型不同于Xen，因为虚拟分片是一条流，且动作是与流相关的，而不是与报文相关的。OpenFlow提供一种简单的报文转发方案，其中网元实施流表查找，转发一条到达的报文。如果没有匹配，则报文被转发到控制器，从而使控制器可在每个节点上设置用来转发相应报文的一条转发规则。OpenFlow协议版本1规定控制器可设置流动作，这样可使在转发报文之前修改一个首部字段。例如，在转发报文到下一个网元之前，转发单元可改变目的地址，将报文转发到一个中间设备。虽然流操作可由OpenFlow容易地加以处理，但诸如报文签名验证等报文级特征不是高效的，原因是它们必须由控制器或一个中间设备执行。

就灵活性而言，OpenFlow的主要劣势是，所有虚拟网络都必须基于相同原语（流表查找、通配符匹配和动作）转发报文，原因是在每个网元中所有虚拟网络共享单个数据平面。另外，Xen向不同虚拟网络提供独立的数据平面。为增加灵活性，OpenFlow提供一个细粒度的转发表，这比Xen所采用的当前TCP/IP所用的表要远较灵活。当前，Xen提供基于IP路由的一个转发表，这意味着转发平面仅基于源—目的IP地址。相比较而言，OpenFlow使用一个N维空间定义每条流，其中N是首部中可用来规定一条流的字段数，如图2.9所示。因此，基于所有的N维或基于一个通配符定义一条流，这将一条流的识别缩小到相关首部字段的一个子集［MCK 08］。使用这样一种转发表的结果是，使报文转发不仅基于目的地址，而且基于其他参数，如应用类型。虽然同样的转发表也可在Xen中实现，但目前还是不可用的。

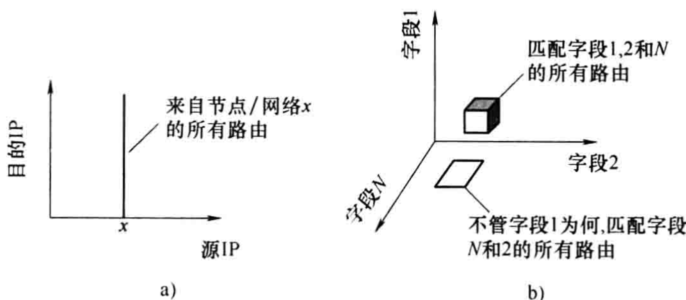


图2.9 在基于TCP/IP网络和基于OpenFlow网络中的转发表

a）基于TCP/IP定义的流 b）基于OpenFlow定义的流

涉及可编程能力的Xen和OpenFlow之间的另一项主要差异是控制平面模型。在Xen中，所有虚拟节点都有数据平面和控制平面。结果，网络控制是去中心化

的。另外，在 OpenFlow 中，每个节点仅有数据平面，控制平面是集中在控制器上的。相比于一种去中心式方法，集中控制平面简化了网络控制算法的开发。但是，一种中心式控制要求在网络中存在一台额外的服务器，这也引入了一个网络故障点。

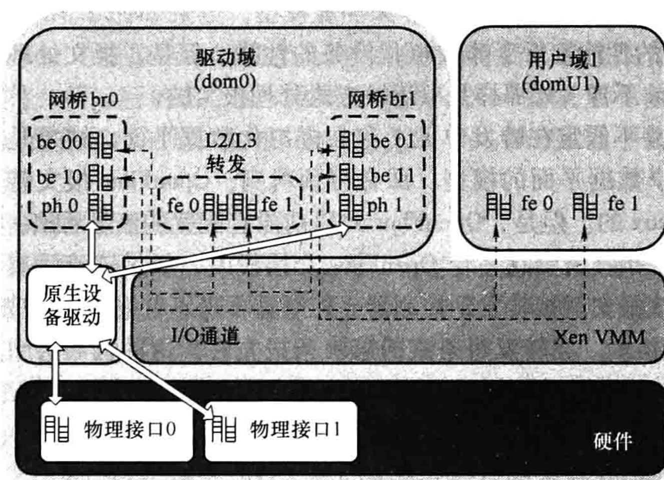
2. 转发性能

多元论架构的一个重要需求是构建在提供高的报文转发性能的一个基层上的。报文转发不仅取决于所用硬件，而且取决于每项技术所提供的逻辑。本节假定，Xen 和 OpenFlow 都运行在相同的硬件上，以便评估每项技术对报文转发所施加的损失。

因为将每个 VM 看作一台虚拟路由器，所以 Xen 作为一台路由器的性能是一个关键点。基本上来说，Xen 的性能取决于报文转发被处理的域。对于每台虚拟路由器，报文转发可由运行在对应于虚拟路由器的运行于 domU 上的操作系统或由 dom0 执行。在第一种情形中，在 dom0 和 domU 之间移动报文诱发控制额外负担，这影响了整体系统性能。在第二种情形中，进出所有虚拟路由器的报文都由 dom0 转发，dom0 同时处理多个转发表。

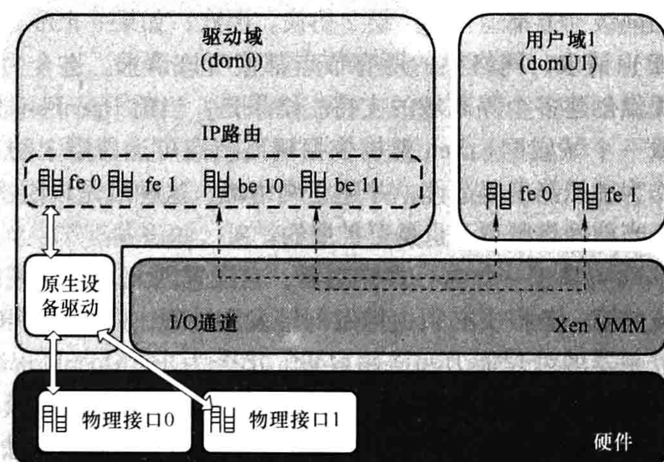
Xen 报文转发的性能也取决于在网络接口之间移动报文所用的模式：网桥或路由模式 [EGI 07]。网桥模式是 Xen 使用的默认网络模式，如图 2.4 所示。不过，这种模式不太适合一台路由器，因为它需要每台设备有一个以上的物理接口。图 2.10a 给出一个带有两个物理接口的网桥模式例子。在 dom0 中有两个网桥，每个物理接口一个，将后台接口和物理接口连接起来。在这种情形中，报文转发可在 dom0 处通过使用层 2 或层 3 协议执行。令 p 是到达物理接口 ph0 的一条报文，它必须被转发到物理接口 ph1。首先，p 由运行在 dom0 上的设备驱动处理。此时，p 处于 ph0 中，它连接到网桥 br0。这个网桥解复用报文 p，并依据目的 MAC 地址，将之移动到后台接口 be00。此后，经过 hypervisor 通过使用 I/O 将 p 从 be00 移动到后台接口 fe0。之后报文 p 被转发到前台接口 fe1，此后，使用另一个 I/O 信道将 p 移动到后台接口 be01。这个接口处在物理接口 ph1 的相同网桥 br1 中。由此，p 到达它的外发接口。值得指出的是，为转发一条报文，hypervisor 被调用了两次。

在如图 2.10b 所示的路由器模式中，dom0 使用物理接口，每个接口关联一个 IP 地址。结果，路由器模式不要求网桥连接每个物理接口和 I/O 信道。因此，在 dom0 处从一个物理接口到另一个接口的报文转发和在原生 Linux 中一样执行。在这种情形中，如果 dom0 被用作一个共享的数据平面（见图 2.3b），则不调用 hypervisor。在路由器模式中，仅当每台路由器实现其自己的数据平面时才调用 hypervisor，如图 2.3c 所示。在这种情形中，报文被路由到与目的 domU 相关联的后台接口，之后经过 hypervisor 通过使用 I/O 通道被移动到前台接口。之后报文被移动到后台接口，最后路由到外发物理接口。为使 domUs 发送和接收报文，也向



fe-前台接口，be-后台接口，ph-物理接口

a)



fe-前台接口，be-后台接口，ph-物理接口

b)

图 2.10 用于报文转发的 Xen 网络架构
a）网桥模式 b）路由器模式

后台接口指派 IP 地址，这与网桥模式不同。

为报文转发使用 VM，支持基于 Xen 的虚拟联网高度可编程能力。每个网络可被配置，并可实施处理以决定在每条所接收报文上采取的动作。但是，如果与一个非虚拟化设备相比，这个特点降低了报文转发性能。为提高报文转发性能，所有必要的规程都可在 dom0 上执行，仅留下控制平面运行在其相应的 VM 上。当报文转发在 dom0 中完成时，对每台虚拟路由器，Xen 有一个控制平面，仅有一个数据平

面为所有虚拟网络共享，这一点不像常规模型，分别如图 2.3b 和图 2.3c 所示。dom0 报文转发的性能接近于非虚拟化设备的性能。但是，报文处理特征失去了灵活性，因为 dom 不像 VM 那样是以逐跳方式处理报文的。

OpenFlow 并不假定在转发单元上存在虚拟化数据平面，结果是，在所有网络间遵循一个共享数据平面的模型。那么可预料到，OpenFlow 报文转发性能是非常类似于原生 Linux 的。但是，OpenFlow 对以前没有配置的流显示出一项劣势。如前面提到的，当一条报文到达一台 OpenFlow 交换机时，如果该流没有匹配转发表上的任何表项，该报文就被转发到控制器。控制器不得不首先配置用来转发报文的所有 OpenFlow 交换机。当转发每条流的第一条报文时，这种机制会引入一个相当大的时延。如果持续地发起新的流，则 OpenFlow 性能就受到影响。

3. 扩展性

扩展性是虚拟化技术的另一个重要问题，目标是提供多个并行网络。虽然相比通过 domU 的 Xen，原生 OpenFlow 具有较佳的报文转发性能，但在网络扩展性方面，Xen 是更有前景的，其中假定考虑的参数为联网单元数。首先，OpenFlow 假定所有节点在相同级别上都运行一个层 2 协议。因此，如果一个节点发送一条地址解析协议（ARP）请求，网络中的所有节点都侦听该请求。迄今为止，OpenFlow 没有在网络中提供创建多个隔离域的支持。结果是，当前 OpenFlow 解决方案受限于一个局域网或一个城域网。Xen 架构依据网络节点可运行层 3 协议的思想。因此，网络节点作为虚拟路由器运作，并建立网络域，这种域是以层次方式组织的。这种结构兼容于当前网络模型，也是可扩展的。

其次，OpenFlow 基于一个中心式控制器，它配置网元。因为控制平面是中心式的，且每条流的第一条报文必须由控制器转发和处理，所以一个 OpenFlow 网络的规模受限于控制器的处理能力和链路容量。迄今为止，OpenFlow 没有为在相同虚拟网络中不同控制器的支持方面提供原生解决方案。同样，Xen 模型给出一种更好的方法，原因是它基于一种去中心式控制平面。虽然去中心式算法会遇到慢收敛且在正确实现方面经常是比较困难的，但当扩展性是一个问题时，它们是比较合适的。

扩展性也与虚拟路由器数有关，这些路由器可运行在相同物理节点之上。未来互联网需求仍然是一个开放问题，且新的架构设计不应该限制运行在可用物理基础设施之上虚拟网络的数量。就这方面而言，Xen 方法是不太灵活的，原因是虚拟网元是一个 VM，相比一个 OpenFlow 交换机中的一条简单流，它要求更多的硬件资源（如处理能力和内存利用率）。在 Xen 中的语境切换和数据路径比 OpenFlow 中要复杂得多。Xen 中虚拟网络数量受限于网元的硬件。实际上，即使一个网元在一个给定时刻没有流量，它也要占据物理网元中固定量的磁盘和内存，这可能阻碍另一个虚拟网元的实例化。OpenFlow 为在一个物理网元之上一个虚拟网络分片的实例化，提供一个更加灵活的基础设施。因为转发网元仅有一个共享的数据平面，其

资源不会被不同虚拟操作系统或为特定虚拟网络维护定量的内存和磁盘所消耗。

因为 OpenFlow 中虚拟网络的概念是由具有相同特征集的流集合给定的，所以 OpenFlow 可支持并行运行的多个虚拟网络。相反，Xen 受限于可在网元硬件上复用的 VM 数量。值得指出的是，如果 dom0 用作一个共享的数据平面，则 Xen 的扩展性可得以改进。

4. 基本虚拟网络管理原语和工具

虚拟联网管理原语取决于如下特定工具：创建和删除虚拟网络，在物理基础设施之上重新分配虚拟网络，重新分配节点资源，以及监测虚拟网络。Xen 和 OpenFlow 给出实现上述原语的不同方法，因为它们基于不同模型，且它们有实现一个虚拟网络的不同方法。

管理虚拟网络会产生在所有虚拟网络管理器之上以层次方式出现的一个实体，在此之后称为裁决器（arbiter）。这是一个重要的假定，原因是如果没有裁决器，则每个网络将尝试消耗所有可用资源，干扰其他虚拟网络的运作。确实，一个虚拟网络管理框架也意味着存在隔离工具，裁决器用之为每个网络保障最小量的资源。在 Feamster 等［FEA 07］的文章中，讨论了存在一个裁决器，在共存虚拟网络间决定可用资源的分割。他们论证说，互联网服务提供商（ISP）应该独立于基础设施提供商，根据他们的说法，基础设施提供商应该负责虚拟网络间的资源共享。

存在一个裁决器提出了有关安全的一个问题。因为有这样一个实体，它控制整个网络，所以这个实体和虚拟/物理节点之间的通信必须是安全的。另外，裁决器不能受到恶意网络节点的影响，该节点想将资源从一个网络转移到另一个网络。在 OpenFlow 中，裁决器自然地定义为网络控制器。如果该架构假定为不同网络使用一个不同控制器，那么这个裁决器就由 FlowVisor［SHE 10］给出。每个网络节点和控制器/FlowVisor 之间的安全信道是在 OpenFlow 标准中定义的。在标准中没有处理访问控制和信任问题，但必须加以实现。Xen 不提供任何类型的裁决器来管理虚拟网络，因为它不是为这种特定的用途开发的。

假定存在一个裁决器，则可讨论在 Xen 和 OpenFlow 中如何针对管理实现基本的虚拟联网操作。在 Xen 中，将一个网络定义为一个虚拟路由器集合。因此，创建和删除一个网络，则分别意味着在相同物理基础设施之上创建和删除虚拟路由器。Xen 为本地实例化 VM 提供机制，其中假定 VM 映像已经存储在物理节点中。因此，为实例化一个虚拟网络，裁决器必须首先选择将使用的虚拟基础设施，将虚拟节点的映像传递到每个物理节点，之后启动 VM。在 OpenFlow 中，一个网络的实例化没有意味着要对转发节点做出改变。实际上，为创建一个新网络，就要求在控制器中实例化一个应用集合，或在使用 FlowVisor 的情形中，要求实例化一个新的控制器。依据到达网络的报文，将应需地创建新的网络流。物理资源（将由每个网络使用）的选择是以在线方式，或在使用 FlowVisor 的情形中在网络实例化的时刻，由控制器确定。Xen 和 OpenFlow 都没有为在相同物理基础设施之上每个共

存的虚拟网络选择最佳配置提供算法。

另一项重要操作是应需虚拟网络调节［PIS 10，CLA 05］，这意味着，如果一个新网络刚被实例化或如果一个运行的虚拟网络的流量发生了变化，要重新分配虚拟网络。在 Xen 中重新分配一个网络可通过路由器迁移、实例化和删除加以实施。如果希望维护相同的虚拟拓扑（这可能是为避免对虚拟网络功能影响的一个重要特点），则迁移是一项必备的解决方案，原因是在不改变虚拟拓扑的条件下，虚拟路由器从一个物理路由器被转移到另一台物理路由器。这样一个规程的约束，是为了确保新的物理路由器至少与原路由器具有相同数量的网络接口。这就使模拟一跳邻居关系的隧道的构建成为必需的，这样的邻居关系在物理基础设施中是不存在的。另外，这可诱发报文丢失，除非使用一些特定的机制［WAN 08］。因此，在 Xen 中实例化和重新分配网络是具有挑战性的操作。但是，OpenFlow 为重新分配网络资源提供一个比较方便的基础设施，因为它仅在每个参与虚拟网络节点上要求流表重新配置。对于控制器这是一项简单操作，原因是它知道物理设备处于哪里以及它们是如何连接的。在 OpenFlow 中，网络重新分配甚至更加简单，原因是存在中心式控制平面，它允许单个单元维护整个拓扑并作用于所有节点。Xen 和 OpenFlow 中网络重新分配的一个例子如图 2.11 所示。

Xen 基于物理节点虚拟化，导致机器部件的资源共享，如内存、磁盘、I/O 访问和中央处理单元（CPU）。实际上，Xen 提供管理资源共享的工具，可被用来分配资源，是否为一些网络提供特权。OpenFlow 提供物理节点的较少控制，原因是网络节点和控制器之间的接口是严格的。因此，在 OpenFlow 中每个网络节点物理资源的控制受限于控制节点和控制器间监测消息的频率，并受限于每个网络所用流表的尺寸。另外，因为控制/FlowVisor 可测量每条流的吞吐量，为控制网络带宽而丢弃一个特定网络的流也是可能的。但是，这些机制没有像 Xen 在控制每个虚拟网络资源那样提供相同的精度。

2.1.4 性能评估

在如图 2.12 所示的由三台机器组成的一个测试床中，评估了 Xen 和 OpenFlow 的性能。流量生成器（TG）通过流量转发器（TF）发送目的地为流量接收方（TR）的报文，TF 模拟一个虚拟网元。TF 机器是一台 HP Proliant DL380 G5 服务器，配备有两个 Intel Xeon E5440 2.83GHz 处理器和 10GB 随机访问内存（RAM）。每个处理器有四个核心，因此，TF 机器同时可运行 8 个逻辑 CPU。在不明确提到时，TF 机器也称作转发器，是采用一个逻辑 CPU 设置建立的。TF 机器使用一个 PCI 快速 x4 Intel 千兆 ET 双端口服务器适配器的两个网络接口。TG 和 TR 也分别称作产生器和接收器，是装备有一个 Intel DP55KG 主板和一个 Intel Core I7 860 2.80GHz 处理器的通用机器。TG 和 TR 机器是通过它们板上的 Intel PRO/1000 PCI（快速网络接口）直接连接到 TF 的。

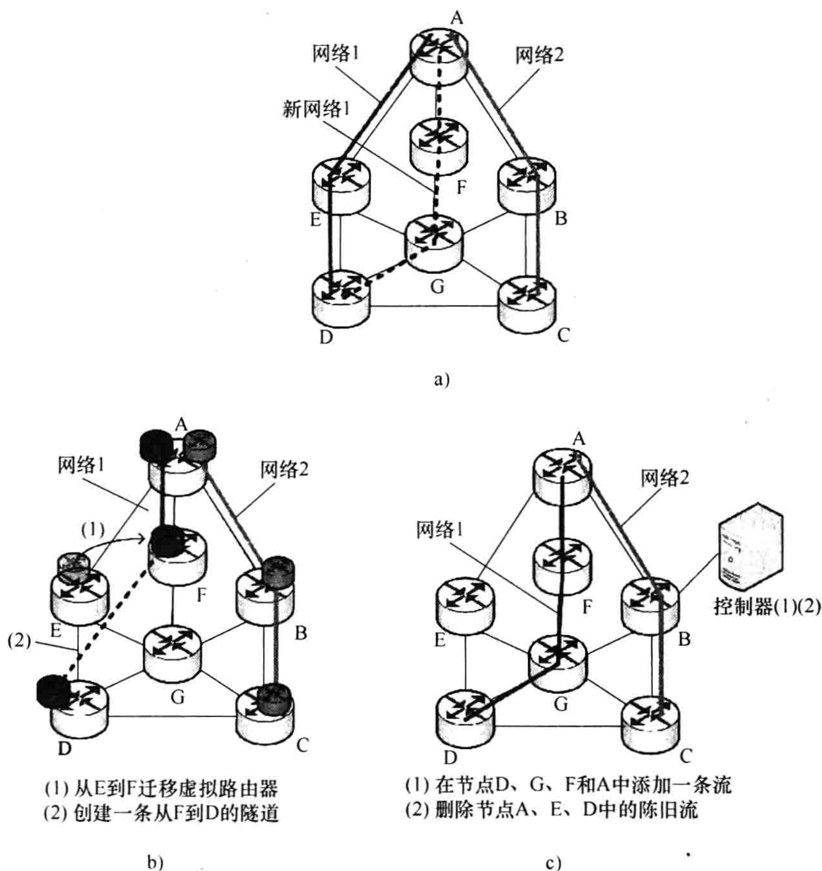


图 2.11 使用 Xen 和 OpenFlow 的网络重新分配例子

a) 原配置 b) 使用 Xen 迁移网络 1 c) 使用 OpenFlow 迁移网络 1

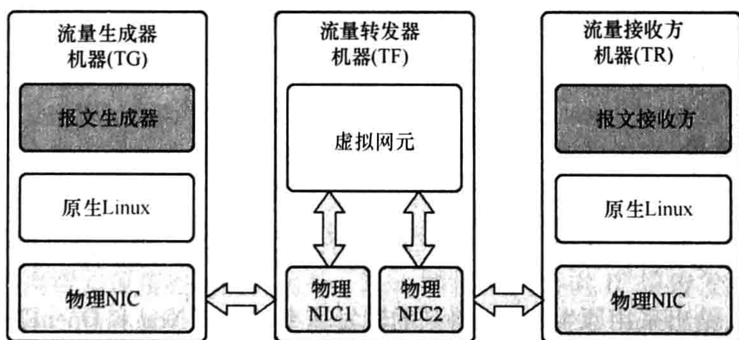


图 2.12 在性能评估中使用的试验场景

在下面的试验中，使用原生 Linux、Xen 和 OpenFlow 评估报文转发性能。使用原生 Linux，转发器运行一个 Debian Linux 内核版本 2.6.26。这个内核也用于 Open-

Flow 试验中，带有一个附加的内核模块支持 OpenFlow。使用 Xen 时，dom0 和 domUs都运行带有一个半虚拟化内核版本 2. 6. 26 的 Debian Linux 系统。

1. Xen、OpenFlow 和原生 Linux 场景

在 Xen 场景中，测试三个不同网络配置。在其中的两种配置中，Xen 运行在网桥模式；在称作 XenVM 的第一种配置中，VM 作为完备的虚拟路由器（在 VM 上有数据平面和控制平面），而在称作 Xen-网桥的第二种配置中，VM 包含控制平面，共享的数据平面运行在 dom0 中。预计 Xen-网桥配置可取得较佳的报文转发性能，即使与 XenVM 配置相比，它降低报文处理灵活性时也是如此。最后，在第三种配置中，Xen 运行在路由器模式，且在这种情形中，它仅评估通过 dom0 的转发报文性能。这种配置被称作 Xen-路由器。在所有配置中使用 Xen hypervisor 版本 3. 4. 2。

在 OpenFlow 场景中，TF 作为一台 OpenFlow 交换机。一台 OpenFlow 控制器使用第三个网络接口连接到 TF。TF 运行 OpenFlow 参考系统版本 0. 8. 9。控制器是一台 IBM T42 笔记本，运行一个 Debian Linux 系统，其上运行 NOX 版本 0. 6. 0 [GUD 08]。使用 pyswitch 应用，它存在于 NOX 中，可在 OpenFlow 交换机中创建流规则。

在原生 Linux 场景中，测试了三个不同的报文转发配置。在第一个配置中，原生—路由器，转发器作为一台路由器，并以静态路由运行标准的 Linux 内核路由机制。原生—路由器配置使用 Linux 内核网桥，它在一台 PC 上实现一台基于软件的交换机。因为层 2 和层 3 解决方案要与 OpenFlow 和 Xen 进行比较，所以需要给出原生 Linux 在网桥和路由器模式下时这些解决方案的性能，以便评估虚拟化对报文转发的影响。但是，在网桥模式中的 Xen 具有与带有网桥的原生 Linux 的一个不同配置。这是因为 Linux 网桥在两个物理接口直接完成层 2 转发，而 Xen 则要进行直到层 3 的转发。为进行网桥模式的 Xen 和原生 Linux 之间的公平比较，为原生 Linux 提供一种混合模式（网桥和路由器），这被称作原生—混合模式。在这种混合模式中，转发器物理网络接口被连接到不同软件网桥，一个核心路由机制在两个网桥之间转发报文。这种配置在原生 Linux 中模拟在 Xen 网桥模式中完成的功能，如图 2. 10a 所示。

2. 试验结果

采用 64B（最小）和 1512B（最大）帧，实施报文转发速率分析。具有 64B 的帧产生高的报文速率，并强制转发器中的高报文处理速度，而 1512B 帧使1Gbit/s物理链路饱和。

图 2. 13a 给出采用原生 Linux 得到的转发速率，这是 Xen 和 OpenFlow 性能的一个上限。也画出点到点报文速率，这是当 TG 和 TR 直接连接时得到的。所得到的低于点到点报文速率的任何速率都是 TG 和 TR 之间丢失报文的后果。结果表明，在路由器模式中的原生 Linux 与直接点到点场景中的性能一样。这由内核路由机制的低复杂性所解释。但是，在网桥模式中，原生 Linux 的性能比路由器模式的要

差。依据 Mateo [MAT 09]，这个结果可能是由于 Linux 网桥实现导致的，为支持高报文速率，该实现没有做优化。最后，观察到混合模式的原生 Linux 给出最差的转发性能。这是预料内的，原因是前面提到的网桥模式限制以及在转发器（TF）中将报文从网桥转发到 IP 层所需要的递增处理开销所导致的。

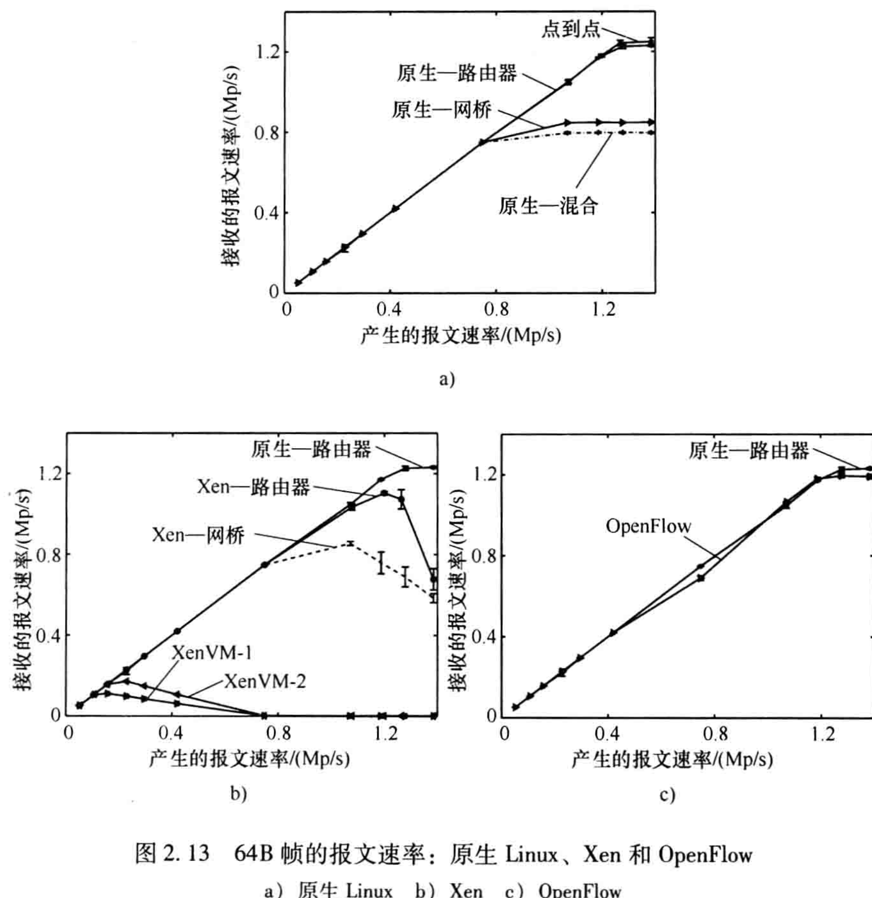


图 2.13 64B 帧的报文速率：原生 Linux、Xen 和 OpenFlow

a）原生 Linux b）Xen c）OpenFlow

Xen 转发速率结果如图 2.13b 所示。首先，分析了 dom0 转发报文的场景。在这个场景中，没有 VM 在运行，虽然如果 VM 处于打开但没有转发报文时，人们期望相同的结果 [EGI 07]。在这个试验中，分析了 Xen 网桥和路由器模式。Xen-网桥使用 Linux 网桥互联各个 VM，但它也遇到原生 Linux 处于网桥模式时的相同限制，原因是网桥实现是相同的。另外，Xen-网桥从网桥向 IP 层转发报文，这和混合模式中一样，与这种模式中所需 hypervisor 调用组合使用。和预料的一样，Xen-网桥的性能比所有原生 Linux 转发方案要差。另外，Xen-路由器的性能比 Xen-网桥的要好，因为当 dom0 转发报文时，没有使用 Linux 网桥，也没有调用 Xen hypervisor。不过，Xen-路由器仍然比原生—路由器的性能要差。在大约 1.2Mp/s 负载时，转发速率快速减少。对于 Xen-路由器和在下面采用 VM 转发的试验中，也观察到

这种行为。这种性能损伤是与 Xen 实现有关的，并需要进一步的调查研究。

也分析了一个 VM 使用 Xen 网桥模式转发流量的一个场景，这是默认 Xen 网络配置。在 XenVM-1 中，VM 和 dom0 共享相同的 CPU 核。相比于前面的结果，这个结果给出性能方面的一次降低，在前者中 dom0 是转发单元。这种不佳的性能似乎是 CPU 资源高度竞争的后果，因为单个 CPU 核是在两个域之间共享的。为消除对 CPU 资源的竞争，完成了采用 XenVM-2 配置的一项试验，如图 2.13b 所示，其中为每个域分配一个独占的核。采用 XenVM-2 试验得到的性能要好于采用 XenVM-1 试验，但它仍然低于 dom0 结果。这可由 VM 报文转发所涉及的高度复杂性所解释。当流量由 VM 转发时，在到达 TR 之前一定会经过一条比较复杂的路径。在接收报文时，是通过到 dom0 内存的直接内存访问（DMA）进行传递的。域 0 之后将报文解复用到其目的地，得到与接收 VM 相关联的一个空闲内存页，将空闲页与包含报文的页交换，之后通知 VM。为发送一条报文，一个 VM 必须发出一条传输请求，带有到内存区（其中报文处于 Xen I/O 环中）的一个索引指针。域 0 之后轮询 I/O 环，并且当它接收到传输请求时，它将索引指针映射到物理页地址，之后将之发送到物理接口［CHI 07］。这种增加的复杂性部分地要求 VM 被用来转发报文的两幅图中得到的低报文速率负责。

图 2.13c 表明，在路由器模式中，OpenFlow 的性能接近于原生 Linux 的。另外，OpenFlow 和 XenVM 之间的比较，显示出灵活性和性能之间的折中。使用 XenVM，则取得更大的灵活性是可能的，因为数据平面和控制平面完全处在每个虚拟网络管理员的控制之下。但是，在 OpenFlow 中，灵活性是较低的，原因是数据平面是在所有虚拟网络间共享的。另外，因为低的处理额外负担，OpenFlow 的性能要优于 XenXM。如果数据平面被迁移到 dom0，则 Xen 性能可得到改进，这在 Xen-路由器和 Xen-网桥结果中已经看到。但是，在这些情形中，定制数据平面的灵活性降低了。

也采用 1470B 数据报文实施了报文转发试验，如图 2.14 所示。对于大型报文，除了 XenVM-1 和 XenVM-2 外，所有转发解决方案都与原生—路由器场景具有相同的行为。因此，在 TF 中没有报文丢失，在这种情形中的瓶颈是 1Gbit/s 链路。不过，对于一个 VM 与 dom0 共享相同内核的 XenVM-1，得到的报文速率是较低的。在 XenVM-2 试验中，一个独占的 CPU 核被分配给每个域，其行为类似于原生—路由器。由此，在 XenVM-1 结果中的性能减少是由各域之间对 CPU 资源的高竞争导致的。

接下来，给出每种类型虚拟网元对流量延迟影响的一个分析。之后介绍了由网元转发的不同速率的背景流量。对于那些速率中的每个速率，一条互联网控制消息协议（ICMP）echo 请求从生成器发送到接收方，为的是依据所产生的背景流量，评估往返时间（RTT）和抖动。通过测量 ICMP 消息中的抖动，深入研究了网元在网络中是插入一个固定时延还是一个变化的时延，这可能影响实时应用。

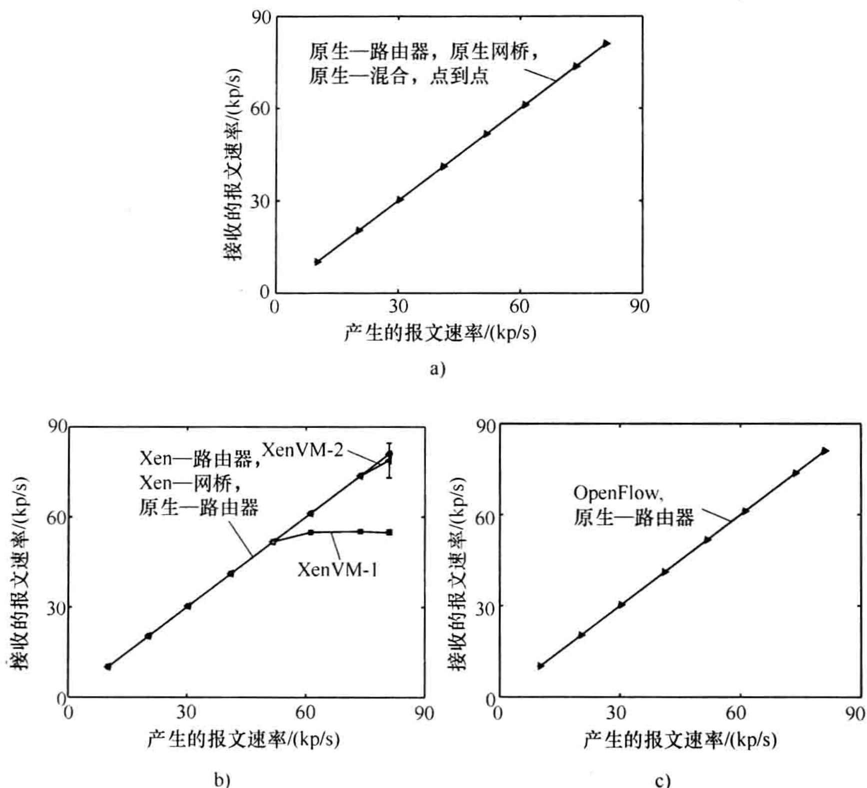


图 2.14　1512B 帧的报文速率：原生 Linux、Xen 和 OpenFlow

a）原生 Linux　b）Xen　c）OpenFlow

图 2.15a 和图 2.15b 分别给出 RTT 和抖动的结果。随着所产生流量的增加，仅针对流量通过 VM（称作 XenVM-1）的配置，ICMP 消息所测得到的 RTT 和抖动是增加的。在最差场景中，XenVM-1 和原生 Linux 试验之间 RTT 的差达到 1.5ms，背景流量是 500Mbit/s。OpenFlow 的 RTT 和抖动与原生 Linux 的 RTT 和抖动，具有相同的量级。不管 XenVM-1 和其他配置之间的时延差是什么样的，XenVMs 在没有对延迟产生显著影响的条件下，可处理网络流量。因为 RTT 总是小于 1.7ms，即使在最差情形中也是如此，所以运行在 Xen 之上的虚拟路由器不会显著地影响实时应用，如 IP 上的话音（VoIP），在不中断通信交互性的情况下，它可容忍高达 150ms 的时延，即使考虑多跳情况时也无例外［FAT 05］。

针对多个网络和每个网络多条流的情况，也分析了 Xen 和 OpenFlow 虚拟化平台的行为。在这个场景中，每个网络被表示为，针对 OpenFlow 是 TG 和 TR 之间的一条报文流，针对 Xen 是一个 VM。报文尺寸和所产生报文速率分别固定在 64B 和 200kp/s。如果有一条以上的并行流，则聚集的所产生流量仍然是一样的。例如，如果采用四条并行流执行试验，则每条流对应于 50kp/s 的报文速率，产生 200kp/s

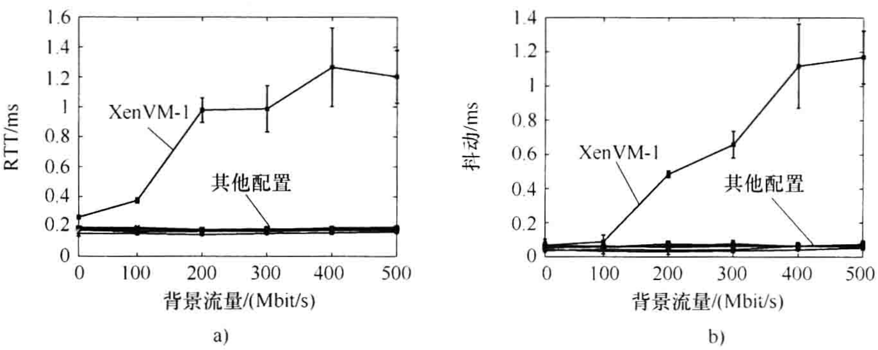


图 2.15 128B 报文的网络时延和抖动

a）往返时间（RTT） b）抖动

的一个聚集速率。图 2.16a 给出作为虚拟网络数的一个函数的聚集报文速率，其中每个网络有一条流。OpenFlow 作为一个软件交换机，除了这样的事实，即流的第一条报文要被发送到 OpenFlow 控制器。得到的性能非常类似于运行于原生 Linux 之上的软件网桥，它维持接近于所产生 200kp/s 速率的接收速率。虽然 Xen dom0 必须使其中断首先由 hypervisor 加以处理，但 Xen-网桥的性能几乎与网桥模式的原生 Linux 一样好。另外，在多个 VM 同时转发流量（XenVM-1 配置）的情形中，随着并行 VM 数增加，性能降低。这个性能降低主要是因为 CPU 语境切换，它必须在逐渐增加的机器数间复用处理器，每台机器乐意转发其自己的流。

图 2.16b 给出聚集的报文速率，它作为流数的一个函数，这里考虑的是单个虚拟网络。和预料的一样，OpenFlow 和 Xen-网桥给出图 2.16a 中一样的行为，原因是两者都有相同的数据平面，结果是，在带有多条流的一个虚拟网络或每个虚拟网络有一条流的多个网络间没有差异。另外，当流量通过 VM（XenVM-1 配置）转发时，在到达 TR 之前，流量一定要经过一条复杂的路径，这在前面的结果中可见到。为验证复杂路径是否为唯一的瓶颈，可在一个配置中重复测试，该配置中 VM

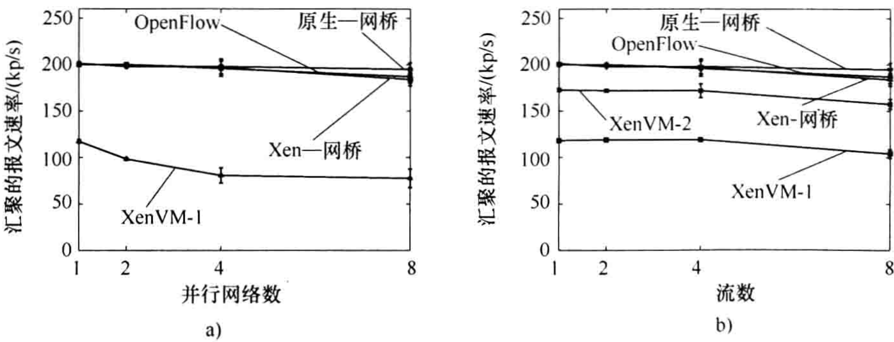


图 2.16 不同虚拟网络数的聚集报文速率

a）网络数的影响 b）流数的影响

没有与 dom0 共享相同的物理核，参见 XenVM-2。在这个配置中，性能增加达到 50kp/s，表明处理能力是网络虚拟化中的一个重要问题。

为分析 CPU 分配对 VM 转发的影响，实施了一项 CPU 变化试验，其中报文从 TG 发送到 TR，以固定速率 200kp/s 通过各个 VM，而分配给 dom0 的专用 CPU 核数是变化的。使用 200kp/s 速率，是因为接近这个速率，就可能在 1-VM 场景中得到最佳性能。依据前面的结果，当 dom0 和 VM 中每个都有一个专用 CPU 核时，转发性能增加。这项测试目标是补充那些结果，方法是当 dom0 独占 CPU 核数增加且有多个 VM 转发报文时，分析转发性能。当使用一个以上的 VM 时，200kp/s 的聚集速率是均等地在 VM 间分配的。图 2.17 给出一个场景中的聚集接收速率，其中每个 VM 有单个核，且专用于 dom0 的 CPU 核数 n 是变化的。依据图 2.17，由于对 CPU 资源的高竞争，所以当所有域共享同一个 CPU 核（即 $n=0$）时，得到最差的性能。和预料的一样，当 $n=1$ 时，性能增加，是因为每个 VM 分配一个专用的 CPU 核，且结果是，就有更多时间执行它的任务。另外，当 dom0 接收一个以上的专用 CPU 核（即 $n \geq 2$）时，dom0 有单个专用 CPU 核，性能就差，甚至当多个 VM 转发报文时也是这样。这些结果表明，当每个虚拟路由器有两个接口时，由 dom0 执行的网络任务是单线程的，且这些任务在一个多核环境中执行得不好。

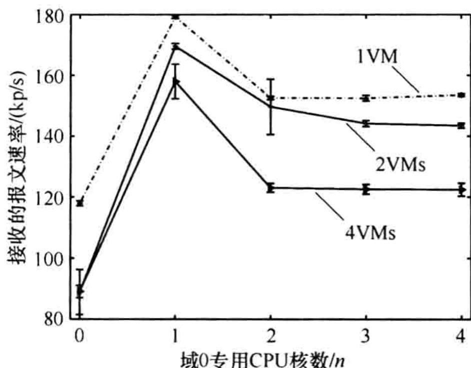


图 2.17　当不同数量的 CPU 分配给域 0 时接收到的报文速率

2.2 Xen 原型

可开发一个 Xen 原型，如电信和自动化集团（Grupo de Teleinformática e Automacão）（GTA）实验室［PIS 11］，对新接口进行试验。本节描述与一个 Xen 原型有关的接口，是依据图 2.18 中描述的架构开发的。这个引导平面向虚拟机服务器（VMS）请求服务（2.2.1 节）。这些服务可与基础设施上的感知或动作相关。VMS 实施所要求的动作，并将应答发回引导平面。为了简化引导平面的实现，由 VMS 提供的接口必须是良好定义的和平台无关的。

可开发一个图形用户界面（GUI），这有助于一个 Xen 的运作（2.2.3 节），支持由一个人操作管理网络。这个界面可替换引导平面，并可访问所有的网络管理任务。

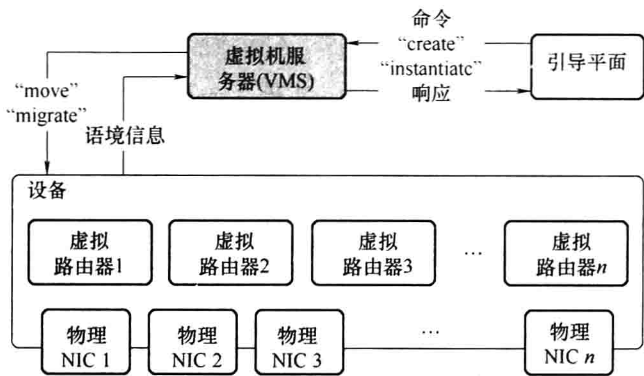


图 2.18 Horizon Xen 原型架构

2.2.1 虚拟机服务器

VMS [ALV 12] 为管理虚拟网络和虚拟路由器提供了一个服务集。为匹配特定需求，这个系统提供了应需虚拟路由器。在接收到新的虚拟网络的请求时，服务器创建合适数量的 VM，并将它们部署到物理网络的特定节点。此外，这个服务器可参与网络管理活动。

该服务器可使用 Web 服务 [W3C] 实现，并使用简单对象访问协议（SOAP）[BOX 00] 对服务请求做出应答。Web 服务器可以是 Apache Tomcat [APA 10]。这种方法简化了 VMS 的异构客户端的生成，并降低了添加新特征的复杂性。每项服务是一个公共方法，建模为称作 VirtualMachineServer 的一个类。

一个引导平面可作用于网络，并可自治地决定实施一些改变，见未来互联网项目中所建议的 [HOR 10]。例如，可减少一个过载物理机器的负载，方法是将其机器中的一个机器迁移到另一个物理主机。在这种情形中，引导平面使用 SOAP 发送一条命令到 VMS，请求迁移一个 VM。之后 VMS 使用 Libvirt [LIB 10]（一个虚拟化系统管理库）实施迁移操作。

在一个 Xen 原型中的可用服务中，存在对管理多个虚拟网络是重要的那些服务。下面针对多元论架构，总结了在一个基于 Xen 的测试床中提供的典型服务，并描述了它们的主要目标。

1）createVirtualMachine：无论何时只要在网络的一个节点上必须创建新的 VM 时，就应该调用这项服务。

2）createVirtualNetwork：这项服务在网络的一些物理节点上创建一个 VM 集。此外，VM 服务器必须将所创建的虚拟网络接口映射到指明的物理网络接口。

3）destroyVirtualMachine：这项服务销毁一个 VM。一个被销毁的 VM 在未来是不能被重用的。

4）registerNodes：最初，VM 服务器是没有网络中物理主机的任何信息的。这

项服务可被用来将现有节点注册到网络中，即将被存储在 VM 服务器处的 VM 的名字、公开密钥和 IP 地址。

5）getPhysicalServerStatus：这项服务得到有关物理服务器的基本信息的一个列表。当前列表包含 CPU 数、核数、RAM 内存大小、空闲 RAM 内存量、主机名以及活跃虚拟域的数量和名字。

6）getRegisteredNodes：这项服务返回在 VM 机器上注册节点的一个列表。

7）getVirtualMachineStatus：这项服务返回有关一个 VM 的基本信息的一个列表。当前列表包含 VM 的名字、当前 RAM 内存大小、可被使用的总 RAM 内存、虚拟 CPU（VCPU）当前数量、VM 可使用的 VCPU 最大数量、使用的 CPU 使用和 VM 的当前状态。

8）migrateVirtualMachine：这项服务将一个 VM 从一台物理主机迁移到同一网络中的另一台物理机器。

9）sanityTest：这项服务是 VM 服务器的一项健康度测试。客户端将一个字符串发送给服务器，服务器发回相同的字符串。

10）shutdownVirtualMachine：这项服务关闭 VM。在这种情形中，该 VM 可在未来再次使用。

11）topologyDiscover：这项服务在物理网络和虚拟网络上创建一个邻接矩阵。这项服务有一个约束。一个节点必须使用服务 registerNodes 注册到该服务器，成为物理拓扑的组成部分，同样必须将其 VM 注册到虚拟拓扑上。

12）getVirtualMachineSchedulerParameters：这项服务查询 hypervisor，查询 CPU 调度器、一个 VM 的各参数。对于信用调度器，参数是权重和上限。

13）setVirtualMachineSchedulerParameters：这项服务为 Hypervisor 设置一个 VM 的 CPU 调度器参数。对于信用调度器，参数是权重和上限。

为使用 VMS 所提供的能力，必须为之开发客户端。VMS 的一个客户端必须创建带有期望服务及其参数的一条 SOAP 消息。

2.2.2　虚拟机服务器客户端

为简化客户端开发，考虑到每个添加到服务器的新服务，可提出一个类。针对每个新类，为产生消息净荷的一个方法被添加到客户端类。在下面，可能的消息带有其相应的净荷（是以一个客户端类实现的），这些消息与 VMS 交互。在 Horizon 项目中建议提出了这些消息［HOR 10］。

1）createVirtualMachinePayload：这种方法创建一个净荷，来请求创建一个 VM，依据寄居新 VM 的物理机器的名字，也依据这个新 VM 的名字、期望 IP 地址和期望的 RAM 大小。该方法返回一条扩展标记语言（XML）消息，表示为类 OMElement的一个对象，带有操作结果，可能是一次成功或失败。

2）createVirtualNetworkPayload：这个方法创建一个净荷，来请求创建一个虚拟

网络，依据的是将寄居新 VM 的物理机器名的一个列表、带有新 VM 名字的一个列表、带有期望 IP 地址的一个列表、带有期望 RAM 内存大小的一个列表和物理网络接口（将被映射到在 VM 上创建的新的虚拟网络接口）的一个列表。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

3）destroyVirtualMachinePayload：这个方法创建一个净荷，来请求销毁一个 VM，依据的是寄居该 VM 的物理机器名字和 VM 的名字。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

4）getPhysicalServerStatusPayload：这个方法创建一个净荷，来得到物理服务器状态，依据的是物理机器名。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败、CPU 数、核数、总的 RAM 内存大小、空闲 RAM 内存量、主机名以及活跃的虚拟域的数量和名字。

5）getRegisteredNodesPayload：这个方法创建一个净荷，来得到注册的节点，没有参数。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果（可能是一次成功或失败）和注册节点的一个列表。

6）getVirtualMachineStatusPayload：这个方法创建一个净荷，来得到 VM 状态，依据的是寄居 VM 的物理机器和 VM 的名字。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果（可能是一次成功或失败）、VM 的名字、当前 RAM 内存大小、可被使用的总 RAM 内存、VCPU 的当前数量、VM 可使用的 VCPU 最大数量、使用的 CPU 时间和 VM 的当前状态。

7）migrateVirtualMachinePayload：这个方法创建一个净荷，来迁移一个 VM，依据的是源物理机器的名字、目的物理机器的名字、VM 的名字和一个字符串（表明该操作是否将是一次实况迁移，即是否在不中断运行于 VM 上的程序的情况下发生迁移）。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

8）registerNodesPayload：这个方法创建注册节点的一个净荷，依据的是要被注册的物理服务器的一个列表。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

9）sanityTestPayload：这个方法创建一个净荷，来请求一次健康度测试，依据的是将被发送到 VM 服务器的一个字符串。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有 VM 服务器接收的字符串，即测试被认为是比较成功的，如果发送和接收字符串是一样的话。

10）shutdownVirtualMachinePayload：这个方法创建一个净荷，来关闭一个 VM，依据的是寄居该 VM 的物理机器的名字和 VM 的名字。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

11）topologyDiscoverPayload：这个方法创建一个净荷，来发现拓扑。该服务没

有参数。该方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果（可能是一次成功或失败）以及物理和虚拟拓扑。

12）getVirtualMachineSchedulerParametersPayload：这个方法创建一个净荷，来得到 VM 调度器参数，依据的是寄居该 VM 的物理机器名字和 VM 的名字。方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果（可能是一次成功或失败）和信用调度器参数（权重和上限）的值。

13）setVirtualMachineSchedulerParametersPayload：这个方法创建一个净荷，来设置 VM 调度器参数，依据的是寄居该 VM 的物理机器的名字、要被影响的 VM 的名字、权重参数的新值和上限参数的新值。方法返回一条 XML 消息，表示为类 OMElement 的一个对象，带有操作结果，可能是一次成功或失败。

2.2.3　图形用户界面

Xen 原型的 GUI，可使用作为 VMS 组成部分而开发的客户端，通过命令行请求可访问 VMS。GUI 使用的所有服务可通过这个 Web 服务通信接口访问。命令行客户端从 GUI 接收服务和所需要的参数。

下面介绍虚拟机服务器以及原型传感器和执行器。

VMS（2.2.1 节）提供控制网络 Xen 机器的一个集成接口。多数管理任务是使用 Libvirt 库［LIB 10］完成的。但是，存在不能仅使用 Libvirt 库就可完成的一些任务，如拓扑发现。对于这些任务，必须开发另外的应用集。

拓扑发现模块得到虚拟的和物理的网络拓扑。另外，为获得有关 CPU、内存和网络有关的数据等信息，需要另一个模块。在没有报文丢失的条件下，迁移一个虚拟路由器也要求一个模块，同样控制虚拟路由器吞吐量也需要一个模块。后者要求一个调度器模块作用于虚拟路由器之 VCPU 的 CAP（上限）。为完全地取得它们的目标，所有这些附加的模块必须从虚拟路由器和物理路由器以及控制器得到数据。因此，为将所有这些模块互联，也需要一个通信模块。这些模块之间的交互关系如图 2.19 所示。

无论何时当 VMS 希望使用这些应用之一时，它就通过命令行传递期望的请求而调用客户端通信模块。如果请求是发往一台物理路由器的，则提供物理路由器 IP 地址。如果请求是发往一台虚拟路由器的，则物理路由器 IP 地址和虚拟路由器 IP 地址都要传递给客户端通信模块。接下来，客户端通信模块采用该请求构造一条 XML 消息，并通过一个套接字将消息发送到运行在物理路由器内部的服务器通信模块。如果请求涉及物理路由器，则服务器通信模块就创建合适应用的一个实例来处理该请求，以所提供的参数调用应用，并将包含应用响应的一个 XML 返回给客户端通信模块。如果请求涉及一台虚拟路由器，那么物理路由器就作为控制器和虚拟路由器之间的一个中介。在这种情形中，服务器通信模块将由客户端通信模块发送的消息解封装，并将之交给代理模块，该代理模块负责将消息发送到适当虚拟

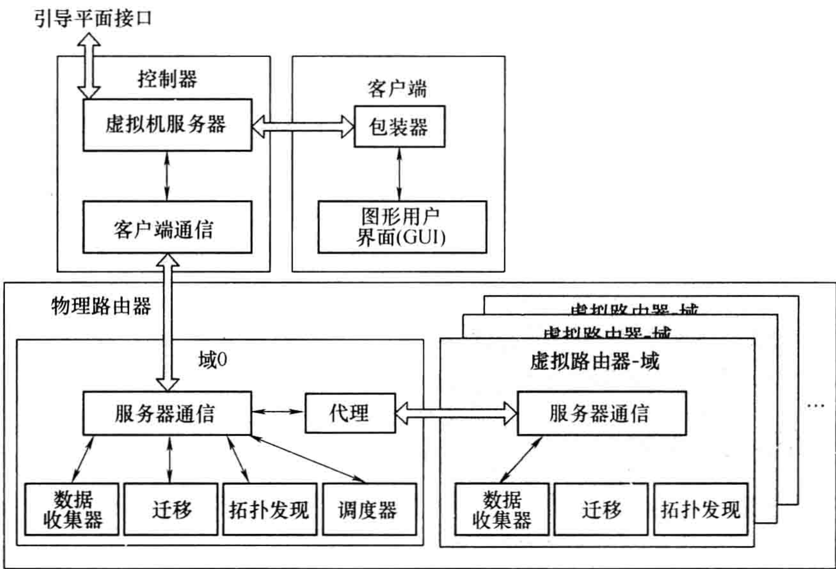


图 2.19　控制器、物理路由器和虚拟路由器模块交互关系

路由器的服务器通信模块。当消息到达虚拟路由器中的服务器通信模块时，就创建适当应用的一个实例，它处理请求，且通过虚拟路由器服务器通信模块、代理模块和物理路由器服务器通信模块，将应用响应发回控制器。

2.3　OpenFlow 原型

在 GTA 实验室［MAT 11］开发了一个 OpenFlow 原型，对新的接口进行试验。本节描述与一个 OpenFlow 原型有关的组件。

遵循 Xen 原型的相同思想，OpenFlow 原型也可基于 Web 服务。核心原型和外部应用之间的通信使用 HTTP（超文本传输协议）来交换 XML 消息。为测量 OpenFlow 网络的性能，可使用传感器。这些传感器基本上是安装在交换机上的计数器（通过 OpenFlow 协议可进行访问）或有关 OpenFlow 表的信息（如流表项数量和其他统计量）。NOX 应用收集传感器信息，并使它们作为一个 Web 服务而为外部使用。

图 2.20 给出 OpenFlow 原型架构。NOX 控制器是 OpenFlow 应用的基础。NOX 控制器为运行于其上的应用提供 OpenFlow 协议和安全信道实现。

2.3.1　应用

下面描述运行于一个 NOX 控制器上的一个应用列表以及它们的主要目标。

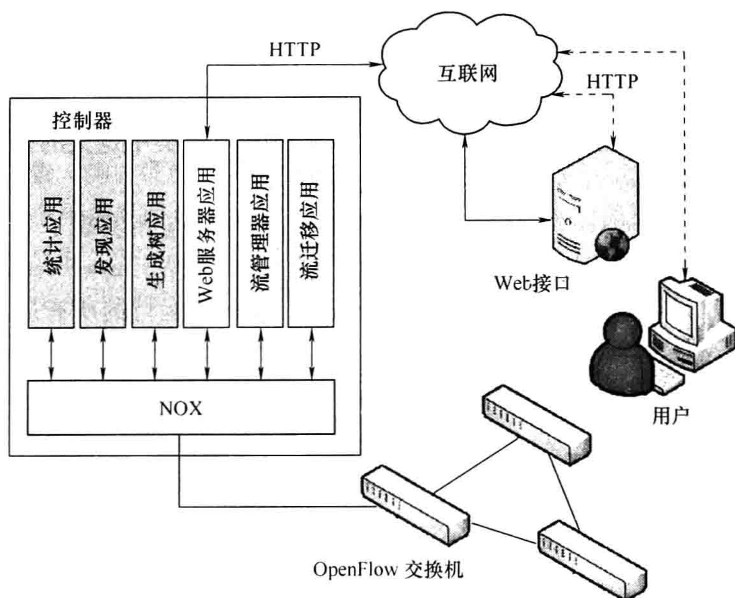


图 2.20 OpenFlow 应用、NOX 以及代理间的交互关系

1）统计应用：这项应用收集有关交换机的统计信息，并将它们转换为一条 XML 消息。

2）发现应用：这项应用发现网络拓扑，并将之描述为一条 XML 消息。

3）生成树应用：这项应用实现一个生成树算法，该算法避免发生网络环路。所定义生成树的拓扑可作为一条 XML 消息。

4）流管理器应用：通过添加、修改和删除流，这项应用实现流变化方面的操作。

5）流迁移应用：通过将一条流从一条路径迁移到另一条路径，这项应用实现流变化方面的操作。

6）Web 服务器应用：这项应用提供 NOX 控制器应用各项特征功能间的集成。WebServer App 实现 HTTP，在 NOX 控制器应用和外部应用之间提供一个接口。结果是，NOX 控制器应用处理 HTTP 请求，将它们转换为一个应用方法调用，并执行该方法。试验所用的原型实现了 WebServer App 的一个客户端。客户端是另一个 Web 服务器，使管理员能够控制 OpenFlow 网络。这个客户端参与到一项 Web 应用，它提供一个 GUI，在后面的 2.3.3 节加以描述。

2.3.2 OpenFlow Web 服务器

WebServer App ［MAT 12］是负责为其他 NOX 应用提供一个 Web 接口的一项 NOX 应用。该 WebServer App 实现 Web 服务的概念，其中其他应用的功能可通过

一条 HTTP 请求加以访问，并返回 XML 消息。这项应用的实现是基于 NOX Web 服务器默认应用的，可被设置运行在8080 端口上，侦听 HTTP 请求。所有 HTTP 请求是由 OpenFlow 资源来处理的，这是在 mywebserver 应用上定义的。对于由 OpenFlow 资源提供的每项 Web 服务，存在在 MyWebServerResource 类中定义的一个方法。下面描述与每个 WebServer App 组件有关的概念。

NOX 默认 WebServer App 为将网站部署为一项 NOX 应用实现了一个框架。这项特征功能被用来将 Web 服务实现为一种特殊种类的网站。mywebserver 类是类实现的一个 NOX 应用。它启动必须与 WebServer App 同时运行的所有应用。它也启动 NOX 默认 WebServer App，并将其默认资源定义为类 MyWebServerResource 的一个对象。MyWebServerResource 类定义一个资源，该资源是一种网站，由 NOX 默认 WebServer App 加以实现。这个类也实现 URL 请求到功能调用的映射，在用户和 OpenFlow 网络之间提供一个接口。存在已经在 MyWebServerResource 上实现的一些服务。每项服务可使用一个特定 URL 由一条 HTTP 请求加以访问。下面描述各项服务。

1）getStats：这项服务不带任何参数。它调用 Stats App（统计应用），并在一条 XML 消息中返回有关 OpenFlow 交换机网络的统计和信息。

2）getTopology：这项服务不带任何参数。它调用 Discovery App（发现应用），并在一条 XML 消息中返回网络的拓扑。这项服务返回所有网络链路的一个列表。

3）getNeighbor：这项服务不带任何参数。它调用 Discovery App（发现应用），并在一条 XML 消息中返回网络的拓扑。这项服务为网络中的每个节点返回所有邻居的列表。

4）getSpanningTree：这项服务不带任何参数。它调用 Discovery App（发现应用），并在一条 XML 消息中返回网络的生成树。这项服务为网络中的每个节点返回节点邻居列表，这些邻居由一条生成树链路连接到该节点。

5）addFlow：这项服务以流特征作为参数，像流匹配、空闲超时、硬超时、优先级和动作等。这项服务添加调用 FlowManager App（流管理器应用）的一条新流，流管理器应用实施在网络上所要求的动作。

6）delFlow：这项服务以流特征作为参数，像流匹配、空闲超时、硬超时、优先级和动作等。这项服务删除调用 FlowManager App（流管理器应用）的一条新流，流管理器应用实施在网络上所要求的动作。

7）migrateFlow：这项服务以流特征作为参数，像流匹配、空闲超时、硬超时、优先级、动作以及在其上必须设置该流的交换机列表等。这项服务调用 FlowManager App（流管理器应用）迁移一条新流，流管理器应用实施在网络上所要求的动作。

2.3.3 图形用户界面

OpenFlow 交换机依据包含活跃流的一个流表，转发网络流量。这个表包含流特征和要实施的规则，如确定队列和输出端口。这个表可以本地方式或由一个网络控制器进行配置。为方便配置和网络管理，可实现一个用户友好的界面，使用户可修改流表。这个用户界面是依据一项 Web 应用开发的，其中使用一个网页浏览器，用户可访问该界面，并运行命令和查询来管理网络。

提供 GUI 的应用被分成三层。第一层是数据层，用于执行用户命令，并实施数据收集作为对查询的响应。第二层是数据处理层，在将所有接收到的信息发送给其他层之前，对之进行处理。第三层是呈现层，它组织数据并将之显示给用户。由层结构提供的隔离，支持在不修改其他层的条件下，修改一个特定的层。图 2.21 给出应用的各层以及用来交换消息的协议。

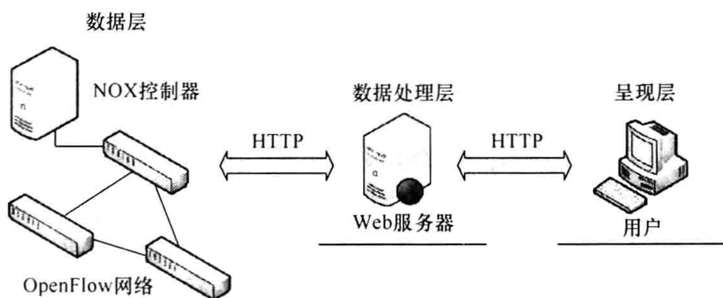


图 2.21　应用层

1）数据层：数据层由一个 NOX 控制器及其各应用组成。应用 WebServer App 负责在数据层和数据处理层之间提供通信接口。这个应用在 HTTP 上与数据处理层通信。

2）数据处理层：数据处理层由一个 Web 服务器组成，它处理来自用户的请求和命令。设计自己的服务器，而不是使用诸如 Apache 的一个现有服务器，对于赋予管理员对所提供服务的完全控制，可能是有用的，这样做的结果是，它有利于系统开发。

3）呈现层：呈现层由标记文件和脚本文件组成，它们由网页浏览器加以解释，可以是超文本标记语言（HTML）、JavaScript（JS）、级联风格表单（CSS）、XML 和可扩展向量图形（SVG）文件。HTML 文件有网页描述，一个网页浏览器可解释之，以便显示这个网页。

CSS 文件将风格进行标记，以改进由 HTML 提供的呈现。一个 CSS 文件由网页浏览器解释，将风格应用到 HTML 标记。JS 文件拥有使网页变得动态和交互性的脚本功能。为显示或交换数据，XML 文件拥有网页浏览器或 JS 函数使用的信息。

作为用户命令响应提供的消息是 XML 消息。

SVG 文件有网络拓扑的一个 XML 描述，以便提供图形可视化。SVG 和 JS 的一个组合支持由 SVG 产生的图像的动画和交互。

在这一层，存在支持用户获取网络信息和执行命令的资源。

2.4 小结

本章使用了两个虚拟化平台 OpenFlow 和 Xen，实施了性能测量。给出的结果表明，Xen 支持一个高度灵活的环境，不同协议栈并行运行，其中使用定制的网络—数据转发结构和查找算法。这种灵活性具有一个高性能代价，将 VM 报文转发容量限制到小于 200kp/s。另外，OpenFlow 给出类似于原生 Linux 环境的报文转发性能。

本章还介绍了每个虚拟化平台的一个原型，同时也介绍了可在每个平台上使用的接口。对于 Xen 平台，使用 Web 服务概念开发了一个 VMS。使用 VMS，可控制网络的物理主机和虚拟主机。为简化一名人类代理的网络管理负担，可开发一个 GUI。这个界面可被用来显示网络的拓扑，并作用于它的各个网元。

在 OpenFlow 原型中，也可开发类似工具。WebServer App 向关注于管理 OpenFlow 网络的各代理提供一个 Web 接口。也可开发一个 GUI。可使用一个网页浏览器访问这个接口，显示与 OpenFlow 网络有关的信息。可向虚拟化平台 Xen 和 OpenFlow 添加更多服务。此外，可提出其他虚拟化平台，并可添加类似服务，且与现有平台进行比较。

2.5 参考文献

[ALV 12] ALVES R.S., CAMPISTA M.E.M., COSTA L.H.M.K., *et al.*, "Towards a pluralist internet using a virtual machine server for network customization", in *Asian Internet Engineering Conference (AINTEC'2012)*, Bangkok, Thailand, pp. 9–16, November 2012.

[AND 05] ANDERSON T., PETERSON L., SHENKER S., *et al.*, "Overcoming the internet impasse through virtualization", *IEEE Computer*, vol. 38, pp. 34–41, April 2005.

[APA 10] "APACHE TOMCAT", March 2010. Available at http://tomcat.apache.org/ (accessed in May 2013).

[BOX 00] BOX D., EHNEBUSKE D., KAKIVAYA G., *et al.*, *Simple Object Access Protocol (SOAP) 1.1*, RFC 3288, March 2000.

[CHI 07] CHISNALL D., *The Definitive Guide to the Xen Hypervisor*, Prentice Hall, 2007.

[CLA 04] CLARK D., BRADEN R., SOLLINS K., *et al.*, New arch: future generation internet architecture, Technical report, USC Information Sciences Institute Computer Networks Division, MIT Laboratory for Computer Science and International Computer Science Institute (ICSI), August 2004.

[CLA 05] CLARK C., FRASER K., HAND S., *et al.*, "Live migration of virtual machines",

Usenix Symposium on Networked Systems Design and Implementation (NSDI), Usenix, pp. 273–286, May 2005.

[EGI 07] EGI N., GREENHALGH A., HANDLEY M., *et al.*, "Evaluating Xen for router virtualization", *International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, USA, pp. 1256–1261, 13–16 August 2007.

[FAT 05] FATHI H., PRASAD R., CHAKRABORTY S., "Mobility management for VoIP in 3G systems: evaluation of low-latency handoff schemes", *IEEE Wireless Communications*, vol. 12, pp. 96–104, April 2005.

[FEA 07] FEAMSTER N., GAO L., REXFORD J., "How to lease the internet in your spare time", *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 61–64, January 2007.

[FER 11] FERNANDES N.C., MOREIRA M.D.D., MORAES I.M., *et al.*, "Virtual networks: isolation, performance, and trends", *ACM SIGCOMM Computer Communication Review*, vol. 66, pp. 339–355, February 2011.

[GUD 08] GUDE N., KOPONEN T., PETTIT J., *et al.*, "NOX: towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.

[HOR 10] "Horizon Project: a new horizon to the Internet", March 2010. Available at http://www.gta.ufrj.br/horizon.

[LIB 10] "Libvirt: the virtualization api", June 2010. Available at http://libvirt.org/.

[MAC 11] MACEDO D.F., MOVAHEDI Z., RUBIO-LOYOLA J., *et al.*, "The AutoI approach for the orchestration of autonomic networks", *Annals of Telecommunications*, vol. 66, pp. 243–255, April 2011.

[MAT 09] MATEO M.P., OpenFlow switching performance, Master's thesis, Politecnico Di Torino, Torino, Italy, July 2009.

[MAT 11] MATTOS D.F., FERNANDES N.C., DA COSTA V.T., *et al.*, "OMNI: Open-flow MaNagement Infrastructure", *IFIP International Conference on the Network of the Future (NoF)*, Paris, France, pp. 1–5, November 2011.

[MAT 12] MATTOS D.M.F., FERNANDES N.C., DA COSTA V.T, *et al.*, "OMNI: OpenFlow MaNagement Infrastructure", in *2nd IFIP International Conference Network of the Future (NoF'2011)*, Paris, France, November 2011.

[MCK 08] MCKEOWN N., ANDERSON T., BALAKRISHNAN H., *et al.*, "OpenFlow: enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, April 2008.

[MEN 06] MENON A., COX A.L., ZWAENEPOEL W., "Optimizing network virtualization in Xen", *USENIX Annual Technical Conference*, Boston, USA, pp. 15–28, May 2006.

[PFA 09] PFAFF B., HELLER B., TALAYCO D., *et al.*, OpenFlow switch specification version 1.0.0 (wire protocol 0x01), Technical report, Stanford University, December 2009.

[PIS 10] PISA P.S., FERNANDES N.C., CARVALHO H.E.T., *et al.*, "Open-Flow and Xen-based virtual network migration", *IFIP International Conference on the Network of the Future Conference (NoF)*, Brisbane, Australia, pp. 170–181, September 2010.

[PIS 11] PISA P.S., COUTO R.S., CARVALHO H.E.T., *et al.*, "VNEXT: virtual NEtwork management for Xen-based Testbeds", *IFIP International Conference Network of the Future (NoF)*, Paris, France, pp. 1–5, November 2011.

[SHE 10] SHERWOOD R., CHAN M., COVINGTON A., *et al.*, "Carving research slices out of your production networks with OpenFlow", *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 129–130, January 2010.

[VER 07] VERDI F.L., MAGALHÃES M.F., MADEIRA E., *et al.*, "Using virtualization to provide interdomain QoS-enabled routing", *Journal of Networks*, vol. 2, pp. 23–32, April 2007.

[WAN 08] WANG Y., KELLER E., BISKEBORN B., *et al.*, "Virtual routers on the move: live router migration as a networkmanagement primitive", *ACM Conference of the Special Interest Group on Data Communication (SIGCOMM 08)*, Seattle, Washington, USA, pp. 231–242, August 2008.

[W3C] W3C *Web Services Activity*, March 2010.

第3章　虚拟网元的性能改进和控制

第2章定义了5个原语（实例化、删除、迁移、监测和设置），这是为支持引导平面控制和管理虚拟网元，网络虚拟化基础设施必须提供的原语［SEN 10］。图3.1给出引导平面和一个通用虚拟化网元之间的关系。

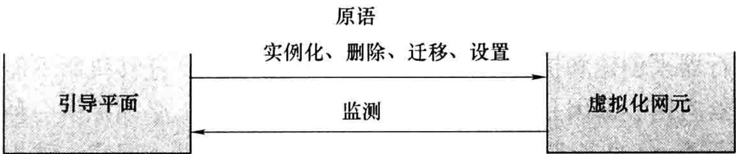


图3.1　控制原语以及引导平面和虚拟化网元之间的关系

基本上，引导平面运行智能算法［SOA 10，FRE 10］，以自动化地实例化/删除虚拟网络，以及迁移网元并设置其资源分配参数［VAN 10，ALK 10］。因此，引导平面需要获取信息，并使用监测原语。这个原语执行到监测工具的调用，这些工具是测量所关注变量（如可用带宽、处理器和内存使用、链路和端到端时延）所需要的。在监测之后，通过使用四个原语，引导平面就能够作用于网络。可用来监测和设置的参数包括低层次和硬件特定的参数，如赋予一个虚拟路由器的虚拟路由器数和在一个竞争场景中处理器使用的优先级。因此，依据当前网络状态、用户数、每个虚拟网络的优先级、服务水平协议（SLA）等，引导平面动态地调整分配给每个虚拟网络的资源。

考虑一个虚拟网元有两个主要平面：虚拟化平面和引导平面。虚拟化平面为在单个共享物理网络之上运行逻辑网元提供基层。引导平面基于SLA为每个虚拟网络进行网络性能优化提供智能。图3.2给出一个通用虚拟网元的架构（如第2章中

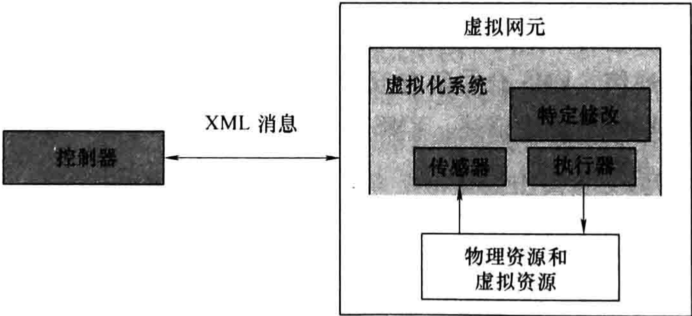


图3.2　一种通用虚拟网元的架构：控制器接收通过传感器获取的数据，并发送命令来管理网元的物理资源和虚拟资源

给出的架构)。节点的核心是虚拟化系统，可由诸如 Xen 或 OpenFlow 等不同虚拟化工具加以实现。传感器和执行器也是虚拟化系统的组成部分。传感器收集支持它们所属语境描述的信息，执行器是实施动作的软件组件，这是在网元上的引导平面所需要的。在图 3.2 中，控制器表示引导平面，它接收和聚集由各节点发送的信息，之后发送命令，设置参数和在各节点上实施控制动作。所有节点必须提供到引导平面的一个接口，并应该能够与之交换可扩展标记语言（XML）控制消息。

本章将焦点放在虚拟化系统所要求的特定修改上面，目的是实现引导平面使用的 5 个原语，并改进虚拟化网络单元的性能。每个虚拟化工具引入一个不同的传感器集合，同时要求不同机制，来管理网络和实现引导平面的各原语。例如，Xen 提供内置的执行器来创建和销毁虚拟机（VM），但原生 VM 迁移机制不能良好地适应于虚拟路由器应用，原因是它没有避免报文丢失。因此，必须提供一种高效的迁移机制。类似于 Xen，OpenFlow 提供创建和销毁流的执行器，但它也缺少一种原生的流迁移机制。因此，针对基于 Xen [PIS 11] 和 OpenFlow [MAT 11] 虚拟化工具，详细描述了实现方面的改进[○]。

3.1 基于 Xen 的原型

基于 Xen 的网络原型由运行在不同类型节点上的几个模块组成。基本而言，每个节点扮演一个不同角色：控制器、路由器或客户端。控制器是一个特殊节点，它接收和合并从所有网络节点接收到的数据。控制器节点获取数据，并将命令发送到物理路由器和虚拟路由器。路由器节点是网络子层。一台物理路由器运行一个或多个虚拟路由器。物理路由器和虚拟路由器运行几个模块，它们实施监测，并在接收到命令时立刻实施动作。最后，客户端节点允许用户使用一个图形用户界面（GUI）与控制器交互。这个 GUI 为用户监测提供原型信息，同时也提供一个简单的控制接口。图 3.3 详细给出基于 Xen 的原型的主要模块及其接口。

控制器有两个模块：虚拟机服务器（VMS）[ALV 12] 和客户端通信。VMS 是控制器的核心。它有一个简单对象访问协议（SOAP）接口，用于与引导平面和客户端节点交互通信。VMS 合并所有原型信息，并执行所有控制和维护算法。在 2.2.1 节中给出 VMS 的详细描述。VMS 使用客户端通信模块，与运行在路由器上的其他路由器节点模块通信。

物理路由器提供由虚拟网络所用的基层。每个物理路由器执行路由器通信模块，它从控制器接收请求，并将这些请求转发到特定模块。如果一条请求的地址是到一个给定虚拟路由器，通过使用代理模块，它被转发到特定虚拟路由器。否则，

○ 这两个原型是由 Horizon 项目组（http://www.gta.ufrj.br/horizon）开发的。

图 3.3　基于 Xen 的原型架构：主模块和相互关系

如果所接收的请求涉及一项监测任务，则它被转发到数据采集器模块从物理路由器和虚拟路由器获取测量数据。如果所接收的请求对应于与虚拟路由器间资源共享有关的一个动作，则调度器模块处理这条请求。拓扑发现模块处理与物理或虚拟网络拓扑发现有关的请求。迁移请求是由迁移模块处理的，在没有报文丢失的条件下，它提供虚拟路由器迁移，这在 3.1.1 节详细描述。

客户端节点使用户能够监测和控制网络原型。这个节点有一个 GUI，显示物理和虚拟网络拓扑，并另外支持细粒度监测，方法是展示有关一台被选中物理路由器或虚拟路由器的详细信息。客户端节点也使用户将命令发送到原型。例如，用户可通过一次鼠标单击，将一台虚拟路由器从一台物理路由器迁移到另一台物理路由器。通过使用包装器模块，运行在客户端上的 GUI 与控制器交互通信。这个模块将命令从 GUI 转换到 SOAP，之后调用控制器。包装器模块也将来自控制器的 SOAP 响应转换到 GUI 所期望的响应模式［KAR 10］。

下面各节描述对在网络原型中采用的 Xen 实现改进。3.1.1 节描述原生 Xen 迁移机制，并介绍我们的机制，这使得可以在没有报文丢失的情况下实现虚拟路由器迁移。3.1.2 节给出进行网络监测的工具。3.1.3 节描述负责发现物理和虚拟网络拓扑的模块。最后，3.1.4 节给出新的虚拟化硬件支持技术［当前它们可用来改进输入/输出（I/O）虚拟化性能］及其对虚拟路由器的影响。

3.1.1　Xen 迁移

迁移原语用来在虚拟路由器间重新分配物理资源。思路是在不中断报文转发和

网络控制服务的条件下，在可用物理资源间移动虚拟网络/路由器［WAN 08］。迁移的一个特例是实况迁移，在不关闭运行路由器的条件下，这允许实施网络拓扑的动态重新配置。因此，实况迁移使物理资源的动态规划和应需流量管理成为可能。

分析标准的 Xen 迁移方案，也提出一种新的迁移模型，该模型修正了在标准机制中发现的问题。迁移是一条重要的控制原语，原因是它使引导平面动态地重新排列逻辑网络拓扑，而不中断运行的服务且没有报文丢失。在这些技术的正式定义之后，给出该原型的一个详细的文档描述，以及原型的概述和功能。

Xen 有一种原生迁移机制，是为移动 VM 而开发的［CLA 05］。这种机制基于两个假设：①迁移是在一个局域网内发生的；②VM 硬盘是在网络之上共享的[○]。因此，VM 迁移由两个主要规程组成：①将 VM 内存复制到新的物理位置；②在不中断连接的条件下，网络链路的重新配置。

实施内存复制，有几种方案。最简单的方案是悬挂 VM，将所有内存页传递到新的物理节点，之后恢复 VM。虽然简单，但这种方案遇到高宕机时间问题，该时间是在迁移过程中一个 VM 不可用的时间。为降低宕机时间，悬挂—传递—恢复规程演化到预复制迁移，它有两个阶段：①迭代预复制；②停止之后复制。在第一个阶段过程中，除了那些称作“热页面”（这是最频繁被修改的页面）内存页外，所有内存页都被传递到新的物理机器。因此，宕机时间得到降低，原因是在 VM 宕机时传递一些页面，仅是热页面而不是所有内存页面。迭代预复制阶段工作过程如下。在第一轮中，以网络管理员指定的一个最小传递速率，将所有内存页从源机器传递到目的机器。之后，在接下来的轮次中，仅有在前一个轮次过程中由操作系统污染的内存页才被传递。依据一种基于“污染速率”的自适应机制，在每个轮次更新传递速率。在每个轮次，计算污染速率，为在上一轮次被污染页数与上一轮次的时长之比。之后，得到下一轮次的最大速率，方法是将 50Mbit/s 的常数增量加到计算得到的污染速率。如果轮次的最大速率等于管理员指定的最大速率或仍然需要传递的污染页小于 256kB，则预复制结束。在此之后，停止之后复制阶段开始，悬挂 VM，并以最大传递速率将热页面传递到目的节点。之后，当目的节点向老的物理节点确认接收到整个内存时，则该过程结束。

清楚的是，对于服务器合并应用，Xen 原生 VM 迁移机制工作良好，但对于虚拟路由器迁移，它却不是高效的。通过使用预复制机制，宕机时间处在数百毫秒，在这个时间过程中的报文会丢失。取决于数据传输速率，在宕机时间过程中，虚拟路由器经历大量报文丢失。为保障快速地释放源物理机器的资源而最小化总迁移时间，同样是重要的。迁移虚拟路由器的 Xen 原生机制的另一个问题是，它假定迁

○ 如果源节点和目的节点实现相同的小型应用集，则这个共享的磁盘假设条件可以松弛［WAN 08］（即不那么严格）。那么，目的节点就能够将这些应用载入到新 VM 的文件系统上。因此，仅有 VM 内存和配置文件是必须被迁移的。

移总是从一台物理路由器迁移到同一局域网（LAN）中的另一台物理路由器。在互联网中，不能假定物理节点总是属于同一个局域网。

为以没有报文丢失地迁移虚拟路由器，一种方案是在 Xen 中实现平面隔离技术［PIS 10］。如在第2章中解释的，一个网元有两个平面：控制平面和数据平面。控制平面运行所有的控制算法并构建路由表，而数据平面转发报文。采用 Xen，两个平面都是在 VM 内（或采用用户域—域 U）内实现的。在原型中，控制平面保持在 VM 中，而数据平面是在域0中实现的。每个虚拟路由器在域0中都有其自己的转发表，且每个表都是原转发表的一个复本，该表是由运行在 VM 中的路由软件构造的。当域0接收一条控制消息时，它检查该消息属于哪个网络，并将该消息转发到相应的 VM。当域0接收一条数据消息时，基于对应于那个虚拟网络的一个转发表，由这个域转发该消息。

采用平面隔离的迁移机制工作过程如下。首先，该规程以与原生迁移机制相同的方式启动：执行迭代预复制阶段，暂停 VM，且剩余的污染内存页被传递到新的物理路由器。在这个时间过程中，数据路径继续在源物理路由器中的域0处工作。因此，直到这个阶段结束之前，既没有中断也没有报文丢失。不同于原生机制的是，零报文丢失修正机制在域0中运行一个守护进程，为正被迁移的 VM 缓冲控制报文。当整个内存都被复制后，在新的物理机器上恢复 VM，使用一个动态接口绑定模块，在这个新的域0中创建网络连接，接口绑定模块将虚拟网络接口映射到新物理路由器的物理网络接口。为传递控制报文（被缓存在源域0中），创建从源物理机器到新物理机器的一条隧道，对于新的控制报文也做如此处理。最后，广播地址解析协议（ARP）应答来更新链路，且在老物理机器中的数据路径被清除。

采用平面隔离的迁移机制保障在 VM 迁移过程中在数据平面没有报文丢失。也没有控制报文丢失。这种机制仅在控制报文交付中插入一个时延。但是，修正机制是基于 Xen 默认迁移的，即它也要求各路由器处在相同 LAN 内。另外，将一条虚拟链路映射到多条物理链路仍然是一个开放问题，它取决于诸如互联网协议（IP）隧道或在网络上实例化新的虚拟路由器等解决方案。例如，在图3.4中，将物理节点2中的虚拟节点 B 迁移到物理节点6。但是，物理节点6不是物理节点1的一个一跳邻居。结果是，为完成链路迁移，需要创建从物理节点6到物理节点1的一条隧道，来模拟一个一跳邻居关系。另一种解决方案是实例化一台新的虚拟路由器来替代隧道。但是，这种解决方案修改了虚拟拓扑，并影响路由协议操作。

为实现平面隔离，在原生 Xen 上的修正如下。首先，为将控制消息发送到合适的 VM，域0应该知道运行在其相同物理机器中的所有 VM。此外，控制平面和数据平面应该交互以维持转发表的一致性。为做到这一点，创建两个组件：Hello 组件和路由改变组件。Hello 组件将每个 VM 的主机名和网络接口信息发送到域0。之后，域0为这台虚拟路由器创建一个路由表和使用这个表的规则。因此，使用虚拟路由器特定路由表，由域0转发这台路由器的所有报文。之后，开始控制平面监

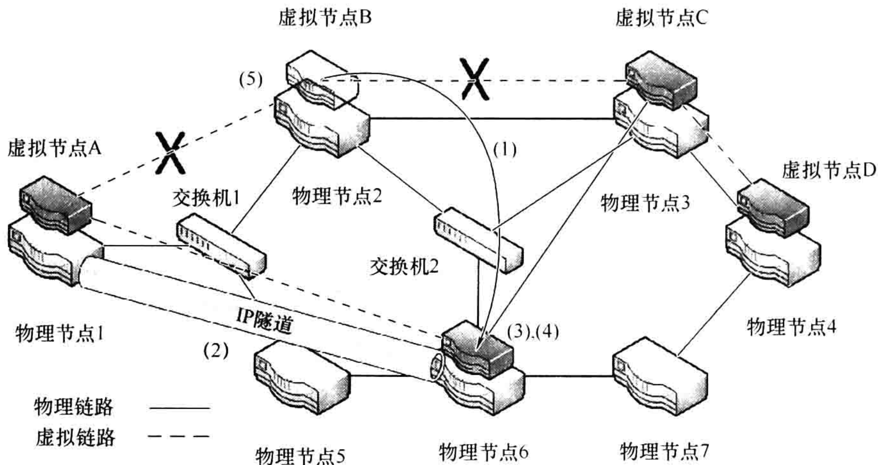


图 3.4 基于 Xen 的路由器迁移例子，其中一条虚拟链路被映射到物理网络中的一条多跳路径（1）将控制平面从物理节点 2 迁移到物理节点 6（维持数据和控制平面）（2）完成接口动态绑定，并在物理节点 1 和物理节点 6 之间创建隧道 （3）在物理节点 6 的域 0 中创建一个新的转发表 （4）以 ARP 应答重新配置链路 （5）删除物理节点 2 上的数据平面

测。当一条路由发生变化时，这个变化也是在域 0 中执行的。路由变化组件监测路由修改。也存在另一个组件，它在迁移过程即迁移建议（Migration Advise）组件过程中使用。这个组件运行在源域 0 和目的域 0 上，并报告迁移过程的成功或失败。使用这个域 0 中的数据平面，路由器继续转发报文，直到网络管理器实际上决定将之迁移到另一台物理机器时为止。标准 Xen 迁移规程是零丢失原型规程的原型第一步。通过使用这种机制，迁移控制平面，但将数据平面保持在其当前物理机器（也称作源域 0）中。在此之后，在目的物理机器（也称作目的域 0）上创建这台虚拟路由器的转发环境、路由表和规则。采用被迁移 VM 中的控制平面路由扩散新的路由表。此时，就使源域 0 转发报文和目的域 0 准备好实施转发。启动链路的迁移。为迁移链路，使用 ARP 机制，这强制原型将一个逻辑跳映射到两个或多个物理跳。因此，迁移发生在与所有逻辑邻居具有连通性的机器之间。为以给定 IP 地址通知接口的介质访问控制（MAC）地址，发送 ARP 应答消息。因此，目的物理机器为使用这台虚拟路由器的每个接口发送 ARP 应答消息，将该路由器当前在另一个位置的信息通知各个邻居。之后，迁移所有链路，并放弃运行在源域 0 之上的数据平面。

3.1.2 Xen 统计信息

为管理虚拟路由器和网络，引导平面要求统计信息。Xen 提供一些工具，获取有关各 VM 的有限信息。因此，开发了测量工具，得到有关虚拟路由器和物理路由器、虚拟网络和网络基层的更一般的信息。这个工具集组成数据采集器模块。基本

上来说，这个模块检索有关域 0 和域 Us 的资源分配和资源使用信息。使用这个信息，原型就可知道整个网络的资源分配状态。

数据采集器模块由几个组件组成，它们专用于从不同测量工具处采集信息，如图 3.5所示。这些组件是 Xentop 采集器、Ifconfig 采集器、内存采集器和延迟采集器。每个组件负责获取一个测量数据集。例如，Xentop 采集器组件采用由 xentop 工具获取的信息，包括域名、中央处理单元（CPU）和内存使用、每个 VM 的虚拟 CPU 数等。Xentop 采集器模块仅在域 0 中执行，并提供来自运行在物理机器内的所有域的非侵入型（non-invasive）信息。数据采集器也有两个特殊组件：数据采集器处理器（负责支持与 Xen 原型的其他模块的通信）和数据采集器主组件（负责调用特定测量模块并合并输出，以满足请求）。由数据采集器提供的各服务可通过 XML 消息请求访问，且应答也是一条 XML 消息。

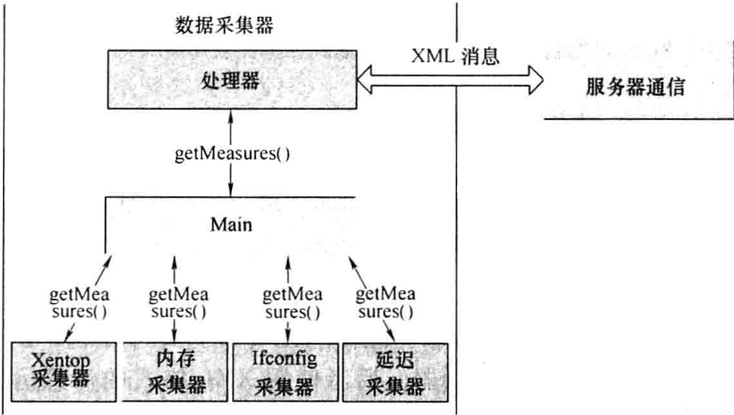


图 3.5　数据采集器架构：主模块将请求发送到特定监测模块，之后将获得的数据转发到处理这些数据的处理器

3.1.3　Xen 拓扑

拓扑模块发现物理网络拓扑和虚拟网络拓扑。这个模块使用 Nmap 安全性扫描器［WOL 02］探测每个网元的所有邻居[1]。拓扑模块有三个组件：扫描邻居、拓扑合并和节点合并。

扫描邻居组件负责发现一个给定网元的所有邻居。物理网元和虚拟网元都运行这个组件。邻居发现使用 Nmap 工具，它探测网元每个网络接口的 IP 范围。这里将对探测做出响应的所有 IP 地址添加到邻居关系列表。此后，就拥有了网元的所有邻居，它们由所有网络接口连接。邻居关系信息有邻居的 IP 地址和 MAC 地址以

[1] 术语“邻居”意指通过网元的网络接口之一有直接相互连接的那些节点。

及链路的延迟。在发现所有网络接口的邻居之后，该组件创建邻居关系列表，并将之通过 XML 消息传递到节点合并组件，如图 3.6 所示。

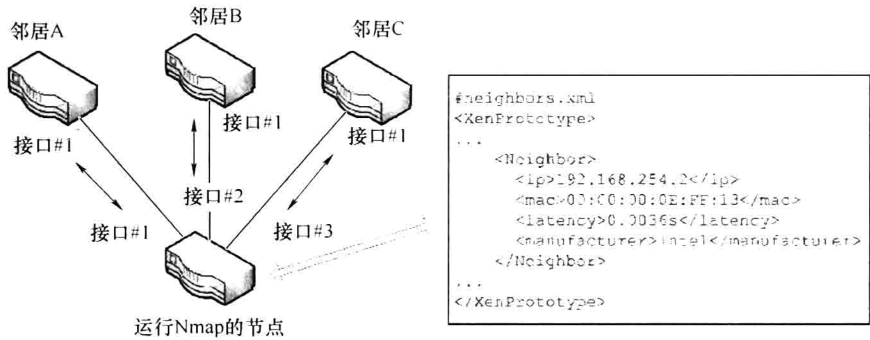


图 3.6 扫描邻居组件：使用 Nmap 工具探测网元的所有接口，由此将对这些探测做出响应的各节点添加到邻居关系列表

节点合并组件运行在网络的每个物理单元上。这个组件目标是得到这个单元在物理网络和虚拟网络中的邻居的信息。拓扑合并组件调用节点合并组件，该组件有三项任务。第一项任务是发现网元的物理邻居。第二项任务是发现运行在这个物理网元之上每个网元的所有邻居。这两个发现规程都使用扫描组件。前两项任务是同时运行的，原因是它们之间没有依赖关系。节点合并组件的最后一项任务是合并信息。合并由物理网元的邻居关系信息和虚拟网元（带有其各邻居）的一个列表组成。该模块将合并信息返回到控制器，后者使用 XML 格式的消息运行拓扑合并组件。

VMS 有物理网络的注册节点的一个列表，但没有有关这些节点间互联关系的信息。拓扑模块提供互联信息。这个模块询问注册节点列表中每个物理节点，就物理单元和虚拟单元的互联信息发出询问。在接收到所有的互联信息之后，拓扑合并组件计算每个网络的拓扑。将网络拓扑建模为一个图。因此，拓扑合并组件将一个图表示返回给 VMS。该图表示为每个网络的邻接矩阵。因此，拓扑模块使 VMS 可提供物理网络和虚拟网络的当前拓扑。

3.1.4 虚拟化硬件改进

虚拟化环境的使用带来性能缺陷，特别对网络虚拟化更是如此。虚拟化层引入额外负担，因为它需要针对 VM 复用实施额外的任务。这项额外负担对 I/O 密集负载是至关重要的 [FER 10]。繁重 I/O 利用率的重要情形之一是网络密集环境。在这种情形中，要求提供高的吞吐量和低时延。此外，每个 VM 的网络流量必须是相互隔离的。

如今，I/O 虚拟化处在虚拟机监测器（VMM）的职责之下，它必须复用外发

数据流和解复用到达数据流。就网络虚拟化而言，VMM 必须共享链路，控制其访问并将到达报文复用到合适的虚拟网络接口。就所有 VM 来说，这些任务必须是公平的。为改进整体网络虚拟化性能，人们已经提出几项技术：设备的直接指派[INT 12]、多队列设备［INT 11b］和单 Root I/O 虚拟化（SR-IOV）［INT 11a］。在下面各节简短地讨论这些技术，也讨论在原型中集成这些新技术所做的当前研究工作。

下面介绍新的 I/O 虚拟化技术。

1）直接 I/O 访问：直接 I/O 技术是由现代主板芯片集提供的一项新功能，安全地支持来自 VM 的直接设备访问。这项技术支持到不同内存区的设备直接内存访问（DMA）。因此，它为一台设备提供将数据直接传递到一个 VM 的能力，而不需要 VMM（或 hypervisor）介入。内存访问由主板的芯片集辅助执行，芯片集截获设备内存访问，并利用 I/O 页表，验证是否允许访问，且将要求的地址翻译到物理内存地址。但是，这个机制具有扩展性问题，原因是一个物理设备是不能在几个 VM 之间共享的。它可被指派到一个 VM。

2）多队列：VMM（或 hypervisor）有分类报文的一项重要任务。报文分类诱发大量 VMM 处理额外负担，因为它要求 VMM 定义所有到达报文的目的 VM，并复用所有外发报文。现代网络接口卡（NIC）解决这个问题的方法是，拥有多个队列，并由队列完成报文分类。为完成这项功能，NIC 使用某个模式［虚拟局域网（VLAN）标签或 MAC 目的地址］对一条报文分类，并将之推入适当的队列。可向 VM 指派一个或多个队列。因此，它们的流量是相互隔离的。

3）SR-IOV：SR-IOV 标准支持在几个 VM 间 PCI 快速设备（如各 NIC）的共享，并好像它们是原生的一样访问这些设备。该标准提供在数据传递中旁路 VMM 介入的一种方式。这种标准方法也为将一个 NIC 共享到几个 VM 定义了一种方法。使用多队列和直接 I/O 的 NIC 访问遵循 SR-IOV 标准。

Xen 版本 4.0.0 具有与新 I/O 虚拟化技术有关的几项新功能。在原型中使用了这个版本的 Xen。一个重要问题是管理域内核，即域 0 内核。需要支持配置设备和 hypervisor 的新功能的一个兼容内核，这使环境具备完全功能。用 Xen 4.0.0 hypervisor 正常工作的最新内核是 2.6.32 Linux 内核（稳定版本）。几项新功能仍然处在开发之下，目前还不支持。当前，为一个 VM 赋予带直接访问的一台设备的完全控制，是可能的。创建 SR-IOV 规格的虚拟功能（用来访问 PCI 快速设备）也是可能的，但将这些虚拟功能的控制赋予一个 VM 是不可能的。

3.2　基于 OpenFlow 的原型

基于 OpenFlow 的原型是建立在运行于 NOX！控制器之上的一个应用集之上的［GUD 08］。NOX 是一个 OpenFlow 控制器，它在 OpenFlow 交换机中配置和管理流。

所开发的 NOX 应用实现 5 个必备原语：实例化流（实例化原语）、删除流（删除原语）、迁移流（迁移原语）、管理和改变流（设置原语）、网络监测和拓扑发现（监测原语）以及在网络中避免环路的一项功能。也存在一个 Web 服务接口，将 5 个原语提供给引导系统。

OpenFlow 原型架构如图 3.7 所示。该原型完成引导系统的原语所定义的传感器和执行器。执行器是流管理器和流迁移应用。流管理器应用提供实例化/删除/设置原语。流管理器应用在其他 NOX 应用和 OpenFlow 命令之间实现一个接口。这个应用负责添加、修改和删除流。流管理器应用接收一条流操作请求，并将之翻译为一条 OpenFlow 命令。另一个执行器是流迁移应用，它实现迁移原语。流迁移在无报文丢失的情况下将一条流从一条路径迁移到另一条路径。统计和发现应用实现原型中的传感器。这两个应用满足监测原语。统计应用测量网络，并收集有关交换机的统计信息。这项发现（Discovery）应用发现网络拓扑，并构造表示网络的一幅图。也存在生成树应用，可避免网络中环路的产生和不必要的广播重传。

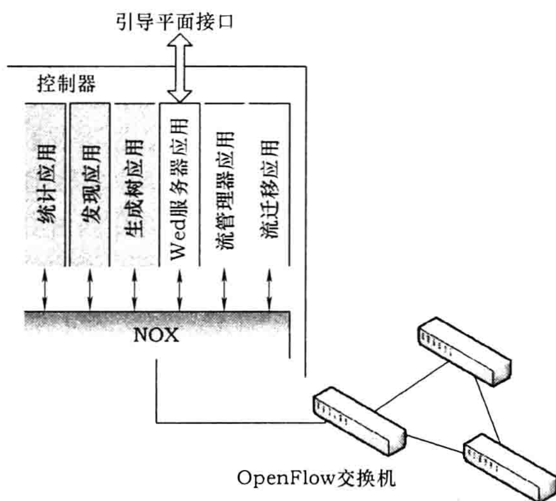


图 3.7　基于 OpenFlow 的原型架构

另外，为将网络分片成几个虚拟网络，我们使用一个工具，它支持几个 NOX 控制器并行运行。称为 FlowVisor [SHE 10] 的这个工具控制网络资源共享，如带宽、控制器看到的拓扑、每个共享的流量、交换机 CPU 使用情况和流表。FlowVisor 实现设置原语，配置每个虚拟网络的参数。

下面各节描述为网络原型开发和修改的各项应用。3.2.1 节给出 FlowVisor 工具，它允许网络资源在 NOX 控制器间的共享。3.2.2 节给出迁移应用，开发该应用是为了以无报文丢失地迁移流路径。3.2.3 节给出统计应用，给出从网络和交换机获取的统计和信息。3.2.4 节描述发现应用，它发现网络拓扑并构造网络的一个图表示。最后，3.2.5 节给出生成树应用，它避免网络中发生环路和不必要的广播消息。

3.2.1　FlowVisor

FlowVisor [SHE 10] 是一种特殊类型的 OpenFlow 控制器。它作为网络设备和其他控制器（如 NOX 控制器）之间的一个透明代理。FlowVisor 的主要功能是对网

络实施分片，并以一种可控的和隔离的方式共享网络资源。

如图 3.8 所示（摘自 Sherwood 等 [SHE 10]），FlowVisor 截获由寄宿（guest）控制器发送的 OpenFlow 消息。之后，FlowVisor 基于用户分片策略（2），透明地修改消息，将到网络的一个分配的控制实施定界。如果消息匹配分片策略，则 FlowVisor 仅将消息从交换机转发到寄宿者（guests）。FlowVisor 对网络分片，保持各分片是相互隔离的。

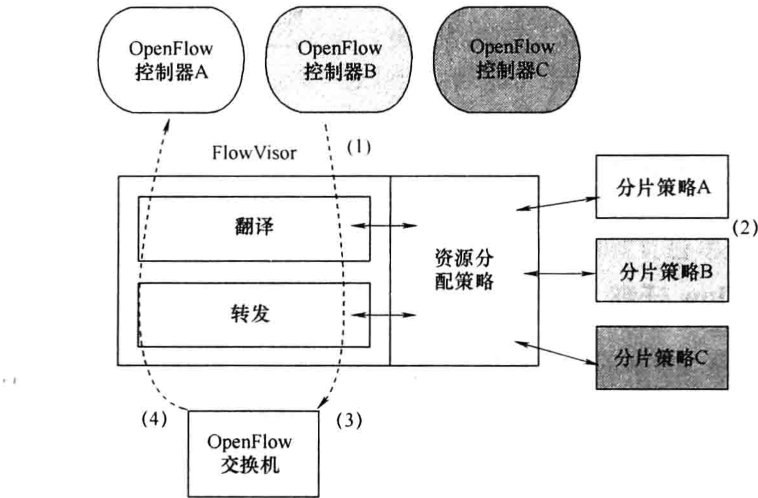


图 3.8　FlowVisor 操作

FlowVisor：

（1）截获由寄宿控制器发送的 OpenFlow 消息　（2）验证用户分片策略　（3）修改消息，对网络的一个分片控制实施定界　（4）将消息从交换机转发到寄宿者，如果消息匹配分片策略

FlowVisor 可对网络实施虚拟化，方法是将交换机的资源分片到几个控制器间。另外，FlowVisor 支持创建 FlowVisors 的层次结构，修改网络架构，如图 3.9 所示（摘自 Sherwood 等 [SHE 10]）。在这种情形中，一个 FlowVisor 实例（FlowVisor 2）并行地虚拟化几个交换机（交换机 1 ~4），且 FlowVisor 1 递归地分片虚拟分片，这由 FlowVisor 2 和 3 确定。为在控制器间将网络分片，FlowVisor 将共享的重点放在 5 项网络资源上：带宽隔离、拓扑发现、流量工程、设备 CPU 监测和转发表控制。例如，带宽隔离是通过在交换机上使用不同优先级队列来保障的。一个队列是与其他队列隔离的。一条流中的所有报文被标记为相同的标识符，之后被映射到 8 个优先级队列之一。基于 3 个不同标识符，对流实施分类：VLAN 优先级码点（PCP）、IP 服务类型（ToS）和 OpenFlow 服务质量（QoS）。默认的是 VLAN PCP。重要的是指出：要保障的是最小带宽。

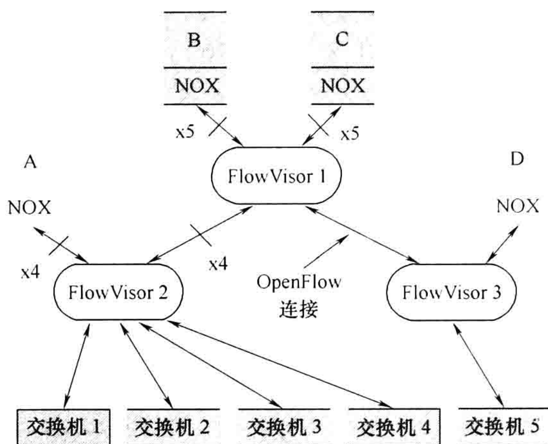


图 3.9 FlowVisor 层次结构：FlowVisor 1 递归地分片由 FlowVisor 2 和 3 定义的虚拟分片

3.2.2 OpenFlow 迁移

流迁移是为原型开发的一项 NOX ［GUD 08］ 应用，它在源和目的 OpenFlow 交换机之间定义一条新的路径，并将流的当前路径改变为新的路径，在这个规程中没有报文丢失。为迁移一条流，流迁移应用以一个交换机列表作为参数。这个列表定义流必须经过的新路径由交换机的一个有序群组组成。为计算从源到目的地（包括选中的交换机）的最短路径，它使用一个广义的 Dijkstra 算法［MOY 98］。

流迁移应用有两个接口：一个是与 Web 服务器应用的接口（通过该接口，改变控制参数），另一个是交换机接口（应用通过它发送流配置命令）。图 3.10 形象地给出流迁移规程的一个例子。最初，源通过路径 1 将报文发送到目的地，该路径由节点 A-F-E-D 组成。定义了从交换机 A 到 D 的一条新流，要求是它必须通过交换机 A、B 和 D，即图中暗灰色节点。A、B 和 D 不是从 A 到 D 的一条完整路径。因此，流迁移应用计算从源到目的交换机的一条完整路径，其中使用 Dijkstra 算法，最小化新路径中的跳数。依据该算法，节点 C 必须被包括在路径中。在定义组成完整路径的所有节点之后，应用在相反方向的交换机中配置该流，即从最接近

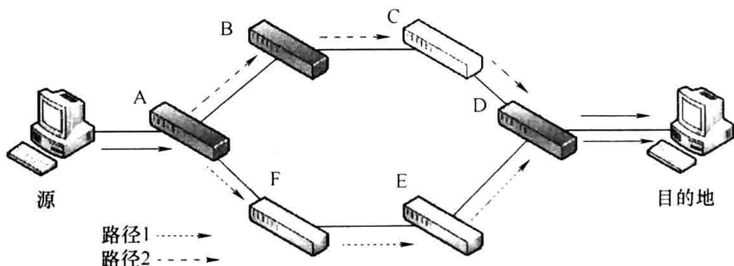


图 3.10 流迁移应用的操作

目的交换机到最远交换机配置该流。因为从目的地到源的所有路径已经配置，例外是源计算机到交换机 A 的链路，这是要被改变的最后一条链路，这样就避免了报文丢失。

3.2.3 OpenFlow 统计

统计应用是另一项 NOX 应用，它是为采集 OpenFlow 交换机信息并将之转换为 XML 消息而开发的。统计应用将命令发送到 OpenFlow 交换式网络，查询每台交换机，查询它的统计和其他信息。每台交换机接收一条请求，将其描述、数据路径、表、汇聚流和流信息报告给 NOX 控制器。采用一条合适的 OpenFlow 协议命令，得到每种信息。统计应用与 Web 服务器和交换机统计应用有接口。Web 服务器应用提供统计应用和一个超文本传输协议（HTTP）客户端之间的一个接口。客户端可以是一个用户界面或另一项应用。交换机统计应用负责将 OpenFlow 命令发送到交换机，并处理响应。出于简单性，将交换机统计应用和统计应用这两者统称为统计应用。

统计应用周期性地从 OpenFlow 交换式网络采集信息。无论何时接收到一条 **stats request**（统计请求），统计应用都以带有当前可用信息的一条 XML 消息做出响应。由统计应用返回的这条 XML 消息包含注册在 NOX 控制器上每台交换机的信息。XML 消息以一条名为 openflowstats 的根标签开始，为每台注册的交换机都有一个 datapath 标签。在每个 datapath 标签内，有交换机标识符号（dpid）、交换机 MAC 地址（dp_mac）和包含从交换机接收到的信息与统计的特定标签。这些特定标签和内部结构存储交换机信息和统计，如交换机 IP 地址（ip）、硬件描述（hw_desc）、交换机实现表数（n_tables）、发送和接收的报文、流计数（flow_count）和流优先级（priority）。

3.2.4 OpenFlow 发现

发现应用是一个修正版本的默认 NOX 应用。首先，使这个应用与任何 OpenFlow 设备兼容，同样为 Web 服务器应用开发了一个接口。发现应用也得到物理网络和虚拟网络的拓扑。

基本上来说，发现应用实现链路层发现协议（LLDP）[IEE 05]。这个协议使各节点能够传输有关网络设备的能力和当前状态的信息。在 IEEE 802 LAN 站的协议栈中，LLDP 实现是可选的。LLDP 帧格式如图 3.11 所示。LLDP 帧将设备信息存储在类型—长度—值（TLV）结构之中。之后，发现应用为一台给定交换机的所有端口创建一个 LLDP 帧。交换机通过其端口传播那些帧。在接到第一条 LLDP 帧时，交换机不知道那条帧的转发规则。交换机将该帧转发给控制器，由之分析帧的内容。在接收到该帧之后，控制器请求发现应用的运行实例处理这条帧。由此，该应用分析是哪台交换机接收到该帧、从哪个端口接收到该帧、是哪台交换机发送该

帧和要从哪个端口发送该帧。采用这个信息，该应用创建带有源交换机、源端口、目的交换机和目的端口的一个数据结构。这个数据结构识别出一条链路以及刻画网络拓扑的一个网络所有链路的聚集。



图 3.11　IEEE 802.3 的 LLDP 帧格式

修改了默认发现 NOX 应用的算法，目的是保障在类型-1 OpenFlow 交换机上拓扑发现规程的正确性。在类型-0 的交换机（OpenFlow 交换机的最简单模型）上，仅当控制器已经为那条帧配置一条流时，该帧才被转发。如果没有配置的流，该帧就被丢弃。但是，即使控制器没有配置特定规则，类型-1 交换机也处理一条流的流量，即默认情况下，初始报文并不被自动地丢弃。这个事实导致网络拓扑的一个不一致表示。

假定从交换机 A 发送到交换机 B 的一条 LLDP。采用默认算法，在处理帧之后，控制器将不会发送丢弃该帧的命令。在这种情形中，B 会不正确地将这条 LLDP 帧转发到连接到 B 的其他交换机。假定交换机 B 和 C 是直接连接的，但 C 没有直接连接到 A。在这种情形中，B 将该 LLDP 帧转发到 C，由此控制器将错误地识别 A 和 C 之间的一条链路。结果是，就假设了一个错误的拓扑。为修正这个问题，修正了默认算法，以便使之总是将丢弃命令发送到各台交换机。这两个算法之间的差异如图 3.12 所示。

图 3.12a 代表默认发现算法。所有交换机都是类型-1 的。在这个例子中：

1）交换机 A 发送一条 LLDP 帧到交换机 B。

2）在接收到这条帧之后，B 将之发送到控制器。之后控制器标示出。

3）在 A 和 B 之间存在一条链路。控制器不发送一条丢弃命令到 B。

4）B 将 LLDP 帧转发到 C。交换机 C 接收该帧。

5）将之发送到控制器。之后，控制器不正确地假定在 A 和 C 之间存在一条链路。这个算法的修正版本如图 3.12b 所示。前两步与默认算法是相同的。区别在于，在此之后，控制器将丢弃命令发送到 B，由此 B 丢弃 LLDP 帧，而不是将之转发给 C。

默认应用仅提供物理网络拓扑，且也需要有关原型中虚拟网络的信息。为取得这个目标，修正的发现应用接收虚拟网络的定义，如一个 VLAN 标识符或一个 IP 范围，之后它提供带有虚拟网络拓扑的一个结构，它由为那个虚拟网络转发流量的所有交换机组成。

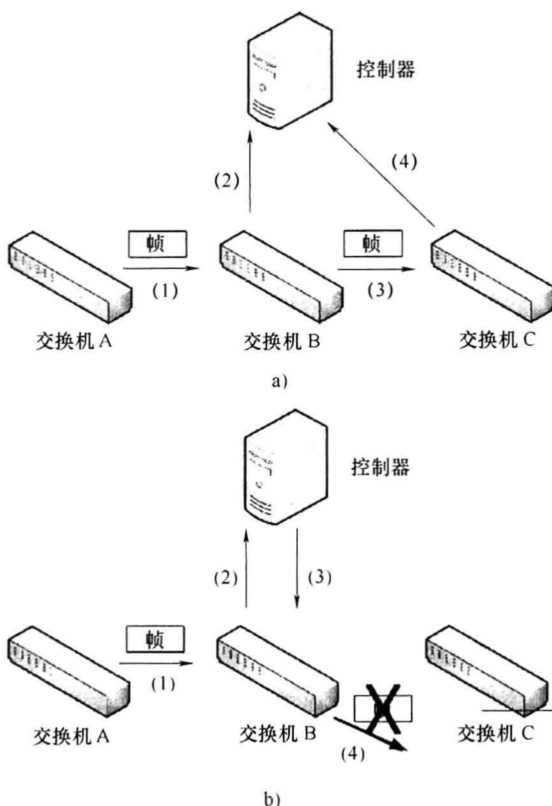


图 3.12 默认和修正发现算法之行为的例子：这两种机制的前两步是相同的，但对于修正算法，控制器将丢弃命令发送到交换机 B（丢弃 LLDP 帧），而不是将之转发到交换机 C

a）默认算法 b）修正算法

3.2.5 OpenFlow 生成树

一个源—目的对之间存在多条路径是不同网络拓扑的一个共性特征。这些冗余路径增加网络可靠性，但在以太网中可导致广播规程中的错误。事实上，如果连接到一台交换机的一个节点广播一个帧，则除了到达端口外，交换机将这个帧转发到所有其他端口。如果网络有冗余路径，则该帧将由所有交换机再次广播，直到它到达源交换机。例如，如果节点 B 发送一条广播消息，则 A 和 C 将接收并转发该消息。当节点 A 接收到由节点 C 转发的该消息时，它将再次转发该消息，且 B 和 C 都将接收并转发该消息。因为交换机 B 不能识别该帧已经被转发，所以它再次转发该帧。因此，由于网络拓扑中的一个环路，则该帧将被不断地转发。使用生成树算法［GIB 11］控制物理拓扑，并在 OpenFlow 原型上避免这些环路。在不产生环路的情况下，这个算法为广播帧生成一个拓扑。

生成树算法作为一个 NOX 应用运行。该算法构造映射网络拓扑的一个图。图的每个节点和每条边分别代表一台交换机和一条链路。该算法分析这个图，并构造包含图的所有节点和被选中边的一棵树，如图 3.13 所示。禁止生成树之外的所有链路（边），意味着这些被禁止的链路将不会转发广播消息。这种机制防止各帧穿越被禁止的链路、发生环路和不必要的广播流量重传。

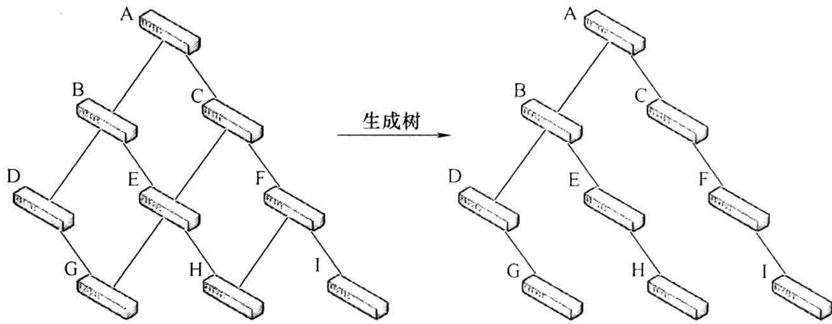


图 3.13 一个网络拓扑及其相应的生成树，假定 A 是生成树的根

生成树算法使用发现应用所提供的拓扑信息工作过程如下。首先，该算法依据交换机（节点）的标识符（Ids）对它们排序。之后，它选择具有最低标识符的交换机作为生成树的根。之后，生成树算法检查连接到根的交换机，并将这些交换机标记为已访问，这意味着该节点以及这些节点和根之间的链路也是生成树的组成部分。最后，它检查连接到已被访问交换机的各交换机。如果这些交换机中的任何交换机被标记为已访问，则这些交换机之间的链路就不构成生成树。否则，该算法将节点和相应链路插入到生成树中。它使用宽度优先搜索算法［ZHO 06］。如果从一个特定节点（如 $n1$）开始，则在搜索距离 $n1$ 的所有 $M+1$ 跳节点之前，宽度优先搜索算法搜索距离 $n1$ 的所有 M 跳节点。例如，在图 3.13 中，如果 $n1 == \mathrm{B}$，则在 G 和 H 之前，该算法搜索 D 和 E。

周期性地运行生成树算法，以保持生成树处于最新状态。将一台新交换机插入到控制器，要求生成树禁止这台交换机的所有端口，直到生成树算法分析图并将新交换机插入到生成树之后才可激活这些端口。出于安全考虑，在一个最小时间量之后，生成树应用激活这些端口以确保它们可由发现应用识别，且结果是，被包括在网络拓扑中。

当一台交换机接收到一条新的广播数据流之后，它将帧转发到控制器。在接收到该帧之后，控制器调用生成树实例分析广播帧被接收的链路。如果这条链路在生成树中，则该交换机处理并转发该帧。否则，应用丢弃该帧。当接收到的帧是一条 LLDP 帧时，处理和转发是独立于源链路发生的，原因是发现应用使用那个帧来发现网络拓扑。同样，如果控制器已经为所接收的被广播帧在交换机中定义一条规则，则交换机不将那个帧转发到控制器，而是依据规则处理该帧，即使链路不在生

成树中也是这样。

3.3 小结

本章描述了在两个虚拟化工具（Xen 和 OpenFlow）上实现引导平面所需 5 个原语（实例化、删除、迁移、监测和设置）的必备改进。描述了两个网络原型。这两个原型都由能够感知和作用于虚拟环境的网元组成。

在基于 Xen 的原型中，采集有关环境信息的模块（如 Xentop 采集器、Ifconfig 采集器、延迟采集器和内存采集器）都是传感器。也存在 Xen 拓扑模块，它们支持物理和虚拟网络拓扑的发现。在基于 OpenFlow 的原型中，传感器能力依赖于所开发的 NOX 应用。统计应用支持控制器检索有关每台 OpenFlow 交换机的状态信息，如穿过交换机的流和每条流接收到的报文数。发现应用发现网络拓扑，生成树应用避免不必要的广播流量重传和网络环路的发生。执行能力支持控制器作用于网络，并实施诸如在线修改单元配置和改变一个虚拟拓扑［在迁移（migrate）原语的辅助下］等任务。在基于 Xen 的原型中，一个经修改的迁移工具支持在无报文丢失的情况下动态重新配置虚拟拓扑。在基于 OpenFlow 的原型中，流迁移应用也可在无报文丢失的情况下修改一条流路径。通过这些执行器，引导平面可动态地重新组织网络拓扑，以确保满足用户的需求。为两个原型开发的这些模块和应用是引导平面开发的基础，这在第 4 章中讨论。

3.4 参考文献

[ALK 10] ALKMIM G.P., DA FONSECA N.L.S., "Virtual network mapping on a physical substrate", *Workshop on Network Virtualization and Intelligence for Future Internet (WNetVirt)*, Búzios, RJ, Brazil, April 2010.

[ALV 12] ALVES R.S., CAMPISTA M.E.M., COSTA L.H.M.K., *et al.*, "Towards a pluralist Internet using a virtual machine server for network customization", in *Asian Internet Engineering Conference (AINTEC'2012)*, Bangkok, Thailand, pp. 9–16, November 2012.

[CLA 05] CLARK C., FRASER K., HAND S., *et al.*, "Live migration of virtual machines", *Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, pp. 273–286, May 2005.

[FER 10] FERRAZ L., CARVALHO H., PISA P., *et al.*, "New I/O virtualization techniques", *Workshop on Network Virtualization and Intelligence for Future Internet (WNetVirt)*, Búzios, RJ, Brazil, April 2010.

[FRE 10] FREITAS R.B., DE PAULA L.B., MADEIRA E., *et al.*, "Using virtual topologies to manage inter-domain qos in next generation networks", *International Journal of Network Management*, vol. 20, no. 3, pp.111–128, 2010.

[GIB 11] GIBB G., "Basic spanning tree". Available at http://www.openflowswitch.org/wk/index.php/Basic_Spanning_Tree, 2011 (accesssed in May 2013).

[GUD 08] GUDE N., KOPONEN T., PETTIT J., *et al.*, "NOX: towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, pp.105–110, July 2008.

[IEE 05] IEEE, 802.1ab IEEE standard for local and metropolitan area networks. Station and media access control connectivity discovery, Technical report, IEEE Institute of Electrical and Electronics Engineers, 2005.

[INT 11a] INTEL LAN ACCESS DIVISION, PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology, 2011, available at http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf, (accessed in May 2–13).

[INT 11b] INTEL CORPORATION, *Intel 82576 Gigabit Ethernet Controller Datasheet*, October 2011. Available at http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82576eb-gigabit-ethernet-controller-datasheet.pdf, (accessed in May 2013).

[INT 12] INTEL CORPORATION, Intel Virtualization Technology for Directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices, 2012 available at http://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices (accessed in May 2013).

[KAR 10] KAROUIA A., LANGAR R., NGUYEN T.-M.-T., *et al.*, "SOA-based approach for the design of the future Internet", *Communication Networks and Services Research Conference (CNSR)*, Montreal, Canada, pp. 361–368, May 2010.

[MAT 11] MATTOS D.M.F., FERNANDES N.C., DA COSTA V.T., *et al.*, "OMNI: OpenFlow MaNagement Infrastructure", in *2nd IFIP International Conference Network of the Future (NoF'2011)*, Paris, France, November 2011.

[MOY 98] MOY J., "OSPF version 2", IETF Network Working Group RFC 2328, April 1998.

[PIS 10] PISA P.S., FERNANDES N.C., CARVALHO H.E.T., *et al.*, "OpenFlow and Xen-based virtual network migration", *World Computer Congress – Network of the Future Conference*, Brisbane, Australia, pp. 170–181, September 2010.

[PIS 11] PISA P.S., COUTO R.S., CARVALHO H.E.T., *et al.*, "VNEXT: Virtual NEtwork management for Xen-based Testbeds", *2nd IFIP International Conference Network of the Future - (NoF'2011)*, Paris, France, November 2011.

[SEN 10] SENNA C.R., BATISTA D.M., MADEIRA E.R.M., *et al.*, "Experiments with virtual network management based on ontology", *Workshop on Network Virtualization and Intelligence for Future Internet (WNetVirt)*, Búzios, RJ, Brazil, April 2010.

[SHE 10] SHERWOOD R., CHAN M., COVINGTON A., *et al.*, "Carving research slices out of your production networks with OpenFlow", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, Búzios, RJ, Brazil, pp. 129–130, April 2010.

[SOA 10] SOARES JR. M.A., MADEIRA E.R.M., "Autonomic management of resources in virtualized networks", *Workshop on Network Virtualization and Intelligence for Future Internet (WNetVirt)*, Búzios, RJ, Brazil, April 2010.

[WOL 02] WOLFGANG M.N., "Host discovery with nmap", November 2002, Available at http://www.nmap.org (accessed in May 2013).

[VAN 10] VANT HOOFT F.N.C., MADEIRA E.R.M., "Resource allocation policies in future multi-agent based virtual network", *Workshop on Network Virtualization and Intelligence for Future Internet (WNetVirt)*, Búzios, RJ, Brazil, April 2010.

[WAN 08] WANG Y., KELLER E., BISKEBORN B., *et al.*, "Virtual routers on the move: live router migration as a networkmanagement primitive", *ACM Special Interest Group on Data Communication Conference (ACM SIGCOMM 08)*, Seattle, Washington, USA, pp. 231–242, 2008.

[ZHO 06] ZHOU R., HANSEN E.A., "Breadth-first heuristic search", *Artificial Intelligence*, vol. 170, nos. 4–5, pp. 385–408, 2006.

第 4 章　语境感知技术的最新状态

Horizon 项目的目标是为后互联网（IP）环境构想并测试一种新的架构。这个后 IP 架构采用网络虚拟化，其中包括一个引导系统。这样一个引导系统的设计，基于多代理系统（MAS），引导系统做出智能决策。

这个项目的主要思路是开发一个环境，其中系统的每个单元贡献信息，用来自动化地更新控制算法，反映环境中的变化，这会影响网络参数的值。为做到这一点，引导系统必须是“自治的”。为支持带有这种自治能力的系统做出决策，调研了有关自治系统 MAS 的最新状态，MAS 支持引导系统和提供这种功能的一些开发平台。

本章描述有关 MAS 和引导系统的最新技术状态。4.1 节描述自治系统的特点。4.2 节介绍 MAS，重点强调它们支持引导系统的特定功能。4.3 节和 4.4 节描述满足项目需求的代理平台，4.5 节给出本章的结论。

4.1　自治系统

一个自治计算系统可被描述为这样一个系统，它感知其操作环境，并对那个环境上系统的行为进行建模。此外，它对发生的变化（无论环境的还是其自己的行为）采取动作。一个目标导向的系统是一个自治系统，它独立地运作，并在没有干预的情况下自己取得它的目标，即使外部环境发生改变也是如此。一个自治系统由自配置、自愈、自优化和自保护的性质组成［IBM 06］。总之，自治系统的主要特点是自治性和自发性。

4.1.1　自治系统的特点

“自治计算”的概念是由 IBM［IBM 06］提出的，它定义了自治计算的四个主要功能：自配置、自愈、自优化和自保护。下面进行描述：

1）自配置是调整自己适应动态变化环境的能力（capacity）。当引入一个独立的组件时，它无缝地集成到其周边环境，系统的其他部分自动地调整自己适应新组件的存在。系统具有自调整的能力，且组件的自动配置遵循高级策略。

2）自愈是发现、诊断并采取动作防止中断的能力。系统必须能够维持它的所有功能，可能处在降级模式，直到发现所有需要的资源（进行恢复）为止。它应该维护有关配置系统的一个知识库，支持诊断推理和分析系统日志（或来自其他系统的日志），以便识别故障。

3）自优化是调整资源和平衡负载的能力，从而可度量资源的使用情况。采用这种方式，各组件应该组织成可提升性能和效率的结构，这要求监测环境、对新选项进行试验并进行学习的能力，以便改进性能优化的选择。

4）自保护是预料、监测、识别和防护威胁的能力。一个自治系统必须监测这些状况，并避免使用的中断。这样的能力要求为所有网元的监测和保护而建立机制和架构。

4.1.2 自治系统的架构和操作

一个自治计算系统由一个连接的自治单元集组成。每个单元必须包括传感器和执行器［STE 03］。系统目标是取得由四个主要功能组成的一个控制环路：采用传感器测量获取的系统性能监测，将传感器测量与预期的比较，如果系统行为不是预期行为而做出决策，以及改变系统行为的动作执行。

1. 自治单元的架构

图4.1给出了一个自治单元的架构［STE 03］。

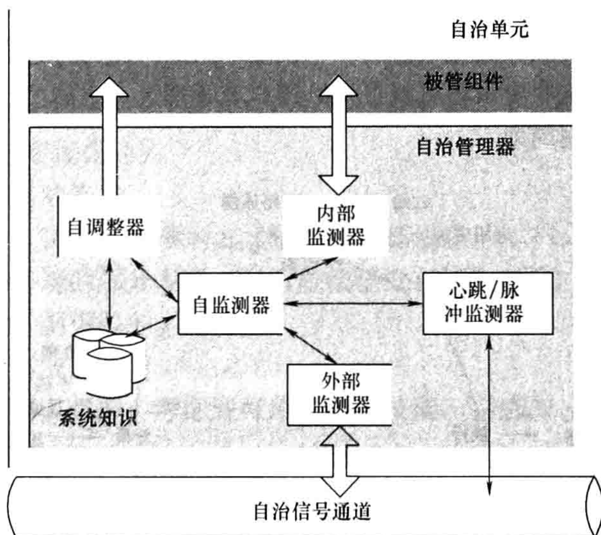


图4.1　一个自治单元的架构

一个自治单元由被管组件和一个自治管理器组成，后者包含几个组件：

1）内部监测器观测被管组件的状态，并将信息传递给自监测组件。

2）外部监测器观测环境的状态，并将这个状态信息从环境传递到自监测组件。

3）自监测器估计被管组件的状态，将之与存储于一个知识库中的期望状态比较，之后得到被管组件之期望状态的偏差。

4）自调整器支持被管组件之状态的调整。

5）心跳监测器汇总自治实体的状态，并将信息传递给负责自治状态控制的其他实体。

图4.1给出由一个被管组件和一个相应的自治管理器组成的自治单元。自治管理器实现所需的自监测和自调整能力。一个内部监测器观测被管组件的状态，并将这个信息传递给自监测器进行评估并采取动作。将测量得到的状态与在一个知识库中保持的期望状态进行比较。将不令人期望的偏差报告给自调整器以便采取动作，这会导致被管组件的改变。类似地，一个外部监测器通过一个自治信号信道观测环境的状态，并可能触发内部变化。信号信道将之连接到其他自治管理器。心跳或脉冲监测器向负责监测状态的其他自治单元提供一个自治单元状态的汇总［STE 03］。

总之，一个自治系统由一个运行在一个控制环路中的自治实体集组成，控制环路确保满足“自我（self）”性质。

2. 自治控制环路

自治系统作为控制环路发挥作用（见图4.2）［DOB 06］。这些系统从各种源收集信息，源包括传统网络传感器和报告流以及较高级设备和用户语境。分析这些信息，构造网络及其服务演化状态的一个模型，从而可做出自适应决策。通过网络执行这些决策，且这些决策将可能报告给用户或管理员。之后可收集这些决策的影响，通知下一个控制周期。

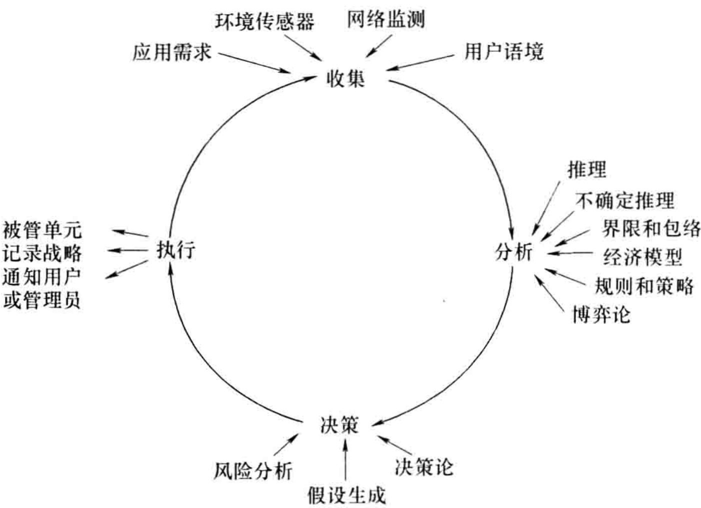


图4.2 自治控制环路

为实现这些系统（包括智能代理），人们提出了几种解决方案。在下一节，给出有关MAS的信息。

4.2　采用多代理系统进行引导

协作自治代理群和电信网络特点之间的对应关系是在电信网络中使用 MAS 的主要动机。MAS 的研究人员搜索具有挑战性的研究案例，而网络共同体搜索一些传统问题的新的解决方案，如服务质量（QoS）提供和避免手工进行网络配置 [BUL 08]。

代理通常被分为两类："认知型" 和 "反应型"；它们之间的区别涉及设计方法及其用途。Ferber [FER 95] 以问题的方式汇总了这种差异：应该将代理理解为已经是智能的实体（能够由它们自己解决某些问题），还是应该将它们比作非常简单的反应式人（可直接对环境变化做出动作）？

4.2.1　代理的定义

不存在 "代理" 概念的被普遍接受的定义。人们提出的定义之一 [FER 95, WOO 02] 是，一个代理是一个物理实体或虚拟实体。它：

1）能够作用于一个环境。

2）可与其他代理直接通信。

3）由一个趋势集所驱动（就其目标，或满意度的一个函数，甚至就生存性而言的，这是其寻求要优化的）。

4）拥有其自己的资源。

5）能够感知（以一种有限的方式感知）其环境。

6）仅有这个环境的部分表示（且可能甚至没有任何表示）。

7）拥有技能并提供服务。

8）可能再生本身。

9）倾向于取得其目标，考虑到可用资源和技能。它感知、呈现其他代理，并与之进行通信。

因此，一个代理可被看作独立于其环境能够思考和动作的一个实体，目标是满足提前设置的目标（由其自己或由一个外部实体设置的）。代理的一些特点如下：

1）泛在性（Ubiquity），是一个基于代理的过程的复杂性和部署能力。

2）互联，在 MAS 的设计中扮演一个至关重要的角色。

3）智能，由任务的复杂度来度量，在没有人类干预的情况下这些任务可自动化或被授权。

4.2.2　代理的特点

一个代理可以是共置的（situated）、自治的、预测式的、反应式的或社交性的，如下所述：

1）共置的：基于从环境接收的感知输入，代理能够作用于其环境。

2）自治的：一个代理应该在没有他人（人类或代理）干预的情况下，能够做出动作，并控制其自己的动作和其内部状态。

3）预测式的：一个代理必须展示出预测式行为，同时能够在正确的时刻采取主动行为。

4）反应式的：一个代理必须能够感知其环境，并在要求的时间内形成响应。

5）社交性的：一个代理必须能够与其他代理（软件或人类）相互作用，实施它们的任务，或辅助其他代理完成其任务。

4.2.3 认知代理

“认知代理”的定义不是唯一的。在参考文献［FER 95］中，存在一个“认知学派”的引用。遵循这个“学派”的研究人员关注于这样的代理，它们可为其行为做出计划。一个认知代理由一个知识库，包括实施其任务以及与其他代理及其环境相互作用必要的所有信息和技能。换句话说，认知代理可被定义为“有意图的”。它们拥有目标和完成这些目标的明确计划。Briot［BRI 01］讨论协商代理，这等价于在参考文献［FER 95］中定义的认知代理。

4.2.4 反应式代理

虽然认知代理可为其行为构造计划，但反应式代理仅具有惯性反应能力。反应式代理被定义为一种特殊类型的代理，认知代理是一般情形。在参考文献［WOO 02］中，反应式代理被定义为不参考其历史而做出反应的那些代理。换句话说，“它们简单地直接对其环境做出反应”。在参考文献［BRI 01］中，作者将反应式代理的架构描述为认知代理的反面。类似于参考文献［WOO 02］中提出的架构，在参考文献［BRI 01］中讨论的架构将反应式代理定义为，仅基于在当前执行时间中收集的信息，做出决策的那些代理。

4.2.5 多代理系统

一个 MAS 由一个计算机进程集组成，它们在某个时间动作，共享共同的资源，并相互通信。MAS 的关键点是代理间协作的形式化（formalization）。涉及代理的主要问题如下：

1）决策。官员决策的机制是什么？代理之间的感知、表示和动作之间的关系是什么？它们如何分解它们的目标和任务？它们如何构造表示？

2）控制。代理之间的关系是什么？它们是如何协作的？这种协作可被描述为完成一项共同任务的协作，或具有不同兴趣的各代理之间的协商。

3）通信。它们相互发送的是什么种类的消息？这些消息遵循哪种语法？取决于代理之间协作的类型，提供不同协议。

MAS 在人工智能领域中有应用，这可降低问题的复杂度，方法是将问题分成

亚组（subgroups）。一个智能代理被关联到每个亚组，且代理之间信息的交换被用来实施协作［FER 95］。这被称作分布式人工智能。在电信中多代理是特别有用的，如电子商务，但也用在其他应用中，如交通系统和机器人技术中的优化。

Ferber［FER 95］将一个 MAS（见图 4.3）定义为由如下部分组成：

1）一个环境 E。

2）一个对象集 O，是无源的，并可与环境 E 中的一个位置关联。它们可由代理创建、收集、修改和销毁。

3）一个代理集 A，代表活跃在系统中的实体。

4）一个关系集 R，在它们之间绑定对象（且因此绑定代理）。

5）一个操作集 O，支持集 A 中的哪个代理能够收集、产生、消费、转换和操作来自 O 的对象。

6）操作员，代表操作的应用和对这种改变尝试的外部世界反应。

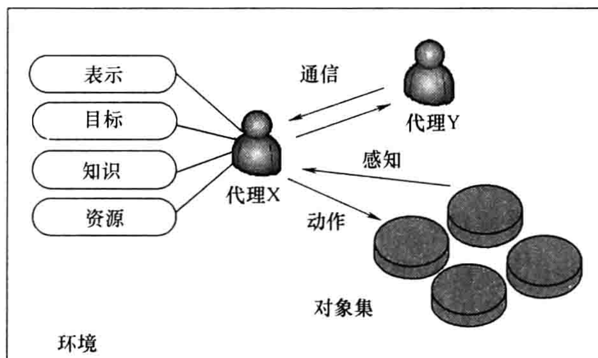


图 4.3　一个多代理系统架构：代理能够感知和作用于一个给定环境的对象，并基于其目标和动机相互作用而实施动作

因此一个 MAS 由运行在一个环境中的几个代理组成，它们相互作用。这些代理能够感知并作用于在这个环境中可被检测到的对象。它们的协作使它们能够依据它们相应的目标和动机而实施动作。一个 MAS 由一个代理集组成，它们在自己间相互作用。每个代理由如下组件组成：

1）收集组件，使代理能够收集有关其环境的信息。

2）相互作用组件，使代理与其他代理相互作用。

3）决策组件，支持代理依据它们的感知做出决策。

4）执行组件，支持代理实施它们的决策。

4.3　构建自治平台的选项

未来的后 IP 架构应该是语境感知的。那么，完成这种功能的平台包括物理的

和逻辑的传感器（软件实体、网络组件和软件代理），收集有关用户和服务的在席状态（presence）、位置、身份和概要的语境信息。典型的语境感知软件涉及服务和用户的本地化，依据用户行为调用服务，为服务组合提供信息，有利于用户间特定通信的进行，以及将 QoS 适配到变化的环境。目标是探索语境感知基础设施的类型，并选择最佳方式将智能引入由一个智能平面和一个引导平面组成的平台。本节描述什么可能是一个自治平台。出于这一点考虑，分析了在 Horizon 项目中可能有用的三个平台：Ginkgo 平台、DimaX 平台和 Java 代理开发框架（JADE）平台。

4.3.1 Ginkgo

Ginkgo 技术为自治联网应用提供支持，它采用分布在网间的网元（NE）中的智能代理。在自治联网应用中，Ginkgo 智能代理扮演一个双重角色：实时地向知识平面提供应用需要的信息，使用知识平面中的分布式知识实时地管理网络控制机制［GIN 08］。

1. Ginkgo 代理和知识平面的共置视图

在 Ginkgo 模型中，知识平面分布在 Ginkgo 代理间，作为一个“共置视图”集。每个共置视图代表一个个体代理的知识，这是就在其邻居关系中的网络位置而言的，其中假定相比在远程位置中的情况，这种位置对代理而言是更重要的。此外，在个体共置视图中更新知识，对于具有有限的控制流量来说，以实时方式完成是比较容易的。一起工作的各 Ginkgo 代理维护整体网络位置的一个总是最新的集体分布式知识，如图 4.4 所示［GIN 08］。

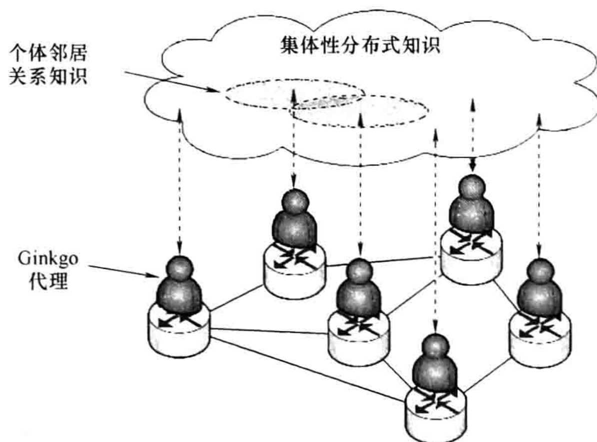


图 4.4 Ginkgo 代理和知识平面

2. Ginkgo 代理架构

Ginkgo 代理由三个主要类型的构造块组成（见图 4.5）：

1）共置视图知识库，专门用来存储各代理共置视图的结构化知识。

2）行为，是自治软件组件永久地将其自己自适应到环境的变化。这些行为中每种行为可被看作具有专家能力的一项专门功能。每种行为本质上是一个感知→决策→动作环路，负责一项控制功能。

3）动态规划器，负责编排行为。动态规划器遵循用户提供的一项策略，指明各行为应该如何考虑到发生在环境中的动态变化。

每种行为（图4.5中的“B”）可执行某项专家功能。行为的典型功能如下：

1）共置视图知识的产生是与其他代理协作产生的。

2）个体性地或集体性地推理，来评估状况，并决定应用一个合适的动作，如一个行为可简单地负责计算网元（NE）的可用带宽。它也可周期性地实施复杂诊断，或它可与特定网络条件的自动识别相关联。

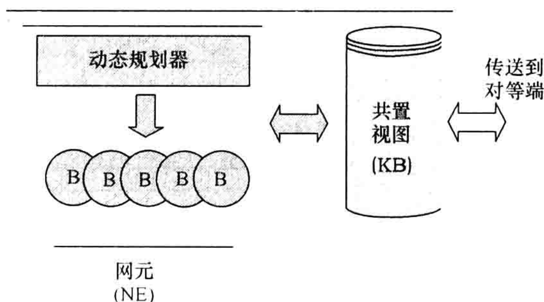


图4.5　Ginkgo代理架构

3）对NE参数实施动作。例如，一个行为可调整一个DiffServ语境中的QoS参数。

4）将有用信息上传到一个网络管理系统。例如，一个行为可上传从基础告警的观测中得到的合成告警。

各行为可访问共置视图，在每个代理内作为一个白板操作，该白板是在代理的行为间共享的。

动态规划器在一个代理内实施行为的激活、动态参数调整和调度。动态规划器确定哪些行为必须是活跃的、何时它们必须是活跃的和采用哪些参数。动态规划器检测共置视图中的变化和外部/内部时间的发生。它编排各代理的反应，以便改变网络环境。为做到这一点，动态规划器遵循一项基于规则的策略，是以一种简单和紧致的形式表示的。

每个代理的共置视图是表示代理之环境的一个结构化知识库。它包含由代理本地收集的知识单元和从其对等端得到的知识。共置视图是在一个周期性基础上更新的，且它被用来将行为自适应到发生在网络中的变化和采取实时决策。一种自治机制将共置视图镜像到合适的对等端；该知识被反映在对等代理的共置视图中。依据知识的本质，可调整这种机制的速率和范围。遵循一个基于本体的模型，组织共置视图，这有助于构建良构的应用，并与其他系统互操作。

4.3.2　DimaX

DimaX是一个容错的多代理平台，它提供像命名、故障检测和恢复等几项服务。为使MAS成为鲁棒的，DimaX使用复制技术。此外，DimaX为开发人员提供

构建 MAS 的可重用组件库。DimaX 提供诸如鲁棒性和可重用能力等有益功能。

1. DimaX 服务

DimaX［FAC 06］是集成一个多代理平台（名为 DIMA）和一个容错框架（名为 DARX）的结果。图 4.6 给出 DimaX 的一个概图及其主要的组件和服务。DimaX 被设计为三层：系统（即 DARX 中间件）、应用（即代理）和监测。在应用层，DIMA 提供构建多代理应用的一个库集合。此外，DARX 将分布和复制代理作为服务所必要的各种机制。DimaX 服务器提供如下服务：命名、故障检测、观测和复制。

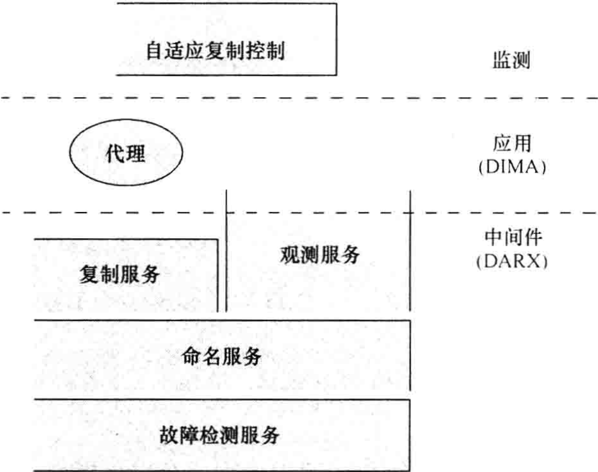


图 4.6　DimaX 架构层次：中间件、应用和监测

命名服务维护在其管理域内所有代理的一个列表（即白页）。当创建一个代理时，将之注册到 DimaX 服务器和命名服务器。故障检测服务（来自 DARX）基于心跳技术：一个进程发送一条“I am alive（我活着）”消息给其他进程，将它是安全的这种信息通知这些进程。当一台服务器检测到另一台 DimaX 服务器的故障时，其命名模块将寄居在故障服务器处所有被复制的代理从列表中清除，并以位于其他主机上的它们的副本替换它们。由故障通知发起替换操作。

对于控制复制而言，观测服务的功能是基础性的。一个观测模块在两个层次收集数据。系统层收集有关 MAS 执行环境的数据（如 CPU 时间和故障间均值时间），而应用层收集有关它的动态特征的信息［如代理间的交互事件（如发送和接收到的消息）］。观测服务依赖于一个反应式代理的组织结构（名为主机监测器）。这些代理收集并处理观测数据，以便计算本地信息，例如在一个给定时段过程中两个代理之间交换的消息数。

DimaX 使用复制机制（复制服务）来避免 MAS 的故障。不管故障为何，复制服务都能够在没有中断的情况下支持 MAS 的运行。一个被复制的代理是这样一个

实体，它在不同主机中拥有其行为的两个或多个备份（或副本）。有两种主要类型的复制协议：主动的和被动的。在主动复制中，所有副本并行地处理所有输入消息，而在被动复制中，仅有副本之一处理所有输入消息，并周期性地将其当前状态传输到其他副本，从而维持了一致性。DimaX 提供几个库和机制，以方便容错 MAS 的设计和实现。下面简短地描述 DimaX 代理的特点。

2. DIMA 代理行为

DIMA 是一个 Java 多代理平台。其内核是预测式组件的一个框架，这些组件代表自治和预测式的实体。一个简单的 DIMA 代理架构由一个预测式代理、一个代理引擎和一个通信组件组成。预测式代理（AgentBehavior 类）代表代理行为。这个预测式组件包括选择合适动作的一个决策组件。例如，一个有限状态机或一个基于规则的系统可被用来描述决策过程。被选中的动作可包括发送消息和一个通信组件，该组件发送和交付消息。提供一个代理引擎，发起和支持代理活动。

3. DarX 任务

DARX 是设计可靠分布式应用的一个框架，这包括一个分布式通信实体集（名为 DarX 任务）。它包括透明的复制管理。在应用处理任务的同时，DARX 处理复制组。这些组中的每个组都由软件实体（副本）组成，它们代表相同的 DarX 任务。在 DARX 中，一个 DarX 任务可被复制数次，且具有不同复制战略。

4. 容错代理

一个容错代理（称作 DimaX 代理）是构建在 DimaX 容错多代理平台之上的一个代理。每个 DimaX 代理具有一个 DarX 任务的结构。但是，DarX 任务不是自治的。为使之成为自治的，将 DIMA 代理行为封装在其内部。这个代理架构支持代理的复制。因为 DARX 中间件和 DIMA 平台为执行控制、通信和命名在不同层次提供机制，所以它们的集成要求一个额外组件的集合。当执行采用 DIMA 开发的多代理应用时，这些组件透明地设置调用 DARX 服务（如复制和命名）。在应用层，要求任意的代码修改。它控制在 DimaX 下构建的代理的执行，且通过 DimaX 服务器，它在远程代理之间提供一个通信接口。

4.3.3　JADE

JADE 是一个软件环境，为联网信息资源的管理构建代理系统，符合可互操作 MAS 的智能物理代理基础（FIPA）规范［FIP 10］。JADE 为基于代理的应用的开发和执行提供一个中间件，这些应用可在有线和无线环境中工作和无缝地互操作。此外，基于一个预定义的可编程和可扩展代理模型以及一个管理和测试工具集，JADE 支持 MAS 的开发［BOR 05］。一个 JADE 环境可动态地演进，原因是依据应用环境的需求，各代理可在系统中出现和消失。对等端之间的通信，不管它们是运行在无线网络还是有线网络中，都是完全对称的，原因是每个对等端都能够扮演发起者和响应者的角色。JADE 是完全采用 Java 开发的，且它基于如下驱动原则

[BEL 03]:

1）互操作性。JADE 符合 FIPA 规范 [FIP 10]。作为结果，JADE 代理可与其他代理互操作，前提条件是它们遵循相同标准。

2）一致性和可移植性。JADE 提供应用编程接口（API）的一个同构集，这独立于底层网络和 Java 版本。实际上，JADE 运行时为 J2EE、J2SE 和 J2ME 环境提供相同 API。原理上来说，应用开发人员在部署时才确定 Java 运行时环境，这是可能的。

3）容易使用。中间件的复杂性是隐藏在一个简单的和直觉的 API 集合背后的。

4）按使用支付的理念。编程人员不需要使用中间件提供的所有功能。没有被使用的功能是不需要编程人员知道它们的任何事情的，且不会添加任何计算上的额外负担。

JADE 向代理编程人员提供如下功能列表:

1）符合 FIPA 的代理平台，它包括代理管理系统（AMS）、默认的目录便利器（facilitator）（DF）和代理通信信道（ACC）。

2）分布式的代理平台。代理平台可被分隔到几台主机上。在每台主机上执行的仅有一个 Java 应用，因此仅有一个 Java 虚拟机。各代理实现为一个 Java 线程，且 Java 事件被用作相同主机上各代理之间有效的和轻量的通信。此外，并行任务可由一个代理执行，且 JADE 以一种协作的方式调度这些任务。

3）为构建多域环境，可在运行时启动多个符合 FIPA 的额外 DF，其中一个域是代理的一个逻辑集合，通过一个共同的便利器通告它们的服务。

4）发送到其他代理/从其他代理接收消息的 Java API；ACL 消息表示为常规 Java 对象。

5）为避免编组和去编组规程，在相同代理平台内 ACL 消息的轻量传输，各消息是作为编码为 Java 对象而不是字符串传递的。

6）管理用户定义本体和内容语言的库。

7）图形用户接口（GUI），管理数个代理和由相同代理构成的代理平台。可监测每个平台的活动，并做日志。通过这个管理性的 GUI，可在代理上实施所有生命周期的操作。

1. JADE 架构

JADE 包括开发应用代理所需的库和提供基本服务的运行时环境，在各代理可被执行之前，这些服务必须在设备上被激活。JADE 运行时的每个实例被称作容器（因为它“包含”代理）。所用容器的集合称作平台（见图 4.7），提供一个同构层，对代理和应用开发人员而言，这隐藏了底层［硬件、操作系统、网络类型和 Java 虚拟机（JVM）］的复杂性和多样性 [BEL 03]。

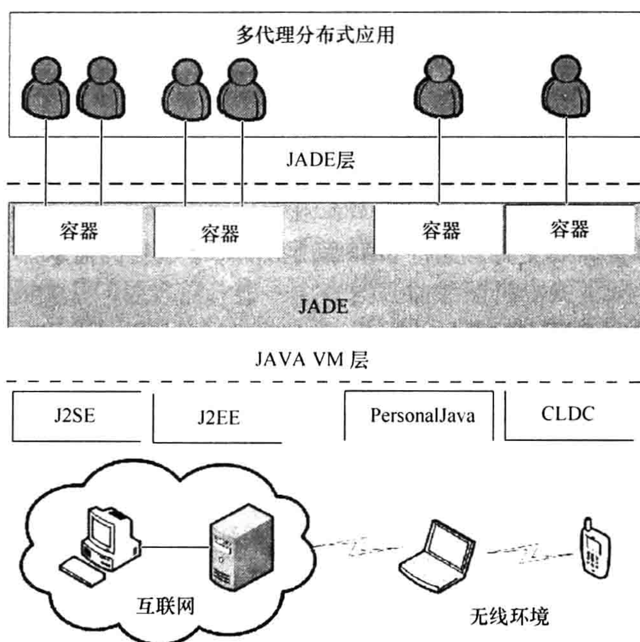


图4.7　JADE 架构模型

2. 构建复杂代理的行为

实现一个代理的开发人员必须扩展 JADE Agent 类，并通过编写一个或多个 Behavior 子类，实现代理特定的任务。用户定义的代理从其超类继承与其平台注册和销户的能力以及一个基本方法集（如发送和接收代理通信语言消息，使用标准交互协议，并注册到几个域）。此外，用户代理从其 Agent 超类继承两个方法：addBehavior（Behavior）和 removeBehavior（Behavior），来管理代理的行为列表［BEL 01］。

JADE 包含代理编程中最常见任务的已定义行为，如发送和接收消息，并将复杂任务结构化为较简单任务的聚集。例如，JADE 提供一个所谓的 JessBehavior，这支持与 JESS［JES 08］的完全集成，JESS 是规则编程的一个脚本式环境，它提供一个使用 Rete 算法处理规则的引擎。

3. 平台管理和监测的 JADE 工具

除了一个运行时库外，JADE 提供管理运行代理平台以及监测和调试代理社会（societies）的工具；所有这些工具都是作为 FIPA 代理本身来实现的，且为实施它们的任务，它们不要求特殊支持［BEL 01］。JADE 工具的例子如下：

1）远程监测代理（RMA）：一个 JADE 代理平台的通用管理控制台。RMA 获取有关平台的信息，并执行 GUI 命令，修改平台的状态。

2）目录便利器（DF）GUI：DF 代理也有一个 GUI，采用它可实施管理，配置它通告过的代理和服务。

3）哑代理是一个简单工具，检查代理间交换的消息，方便代理消息交换模式的验证和一个代理的交互式测试。

4）嗅探代理使我们可跟踪在一个 JADE 代理平台中交换的消息：跟踪来去一个被选中代理或组的每条消息，并显示在嗅探器窗口中，使用类似于统一建模语言（UML）序列图的一种表示。

JADE 是一个有益的选项，它通过 FIPA 规范提供互操作性。它为 MAS 的开发提供一种友好的和鲁棒的支持，这些 MAS 满足 Horizon 项目的需求。

4.4 网络控制的语境感知技术

当前互联网架构有一个唯一的协议栈，传输控制协议/互联网协议（TCP/IP）栈，运行在物理基层之上。因此，依据没有区分性的单个模型，转发来自所有应用（每个应用有不同性能需求）的报文。为向未来互联网中的不同应用提供定制化服务，Horizon 项目建议一种多元论架构，这使不同协议栈同时运行在相同的共享物理基层之上。所提出的架构是基于网络虚拟化概念的［AND 05］。其思路是在相同基层上运行多个虚拟网络，为每个网络提供资源［HE 08］，由此，依据运行在每个虚拟网络上的应用需求，能够创建不同网络。

对于所提供架构，底层网络资源的高效共享是一项基础挑战。在 Horizon 项目中，开发一个引导系统所需的基础设施，来处理不同虚拟网络间的资源分配问题。目标是将来自基础设施的所需信息投射到（project）到引导系统上。基于这种信息，引导系统获取虚拟网络创建和销毁以及物理资源的分布等方面优化所需的知识。对于每个网元（如路由器和交换机），引导系统定义一个共置视图，用来确定单元周边的语境，并选择和优化它们的控制算法和参数。主要思路是定义本地算法（如路由和 QoS 机制），作为语境的一项功能，目的是增加网络的扩展性。因此，引导系统必须感知到语境。

语境感知的系统被定义为这样的系统，它们可感知实体的状况或其语境，并作用于这些实体［LOK 06］。依据这个定义，语境和状况是紧密相关的概念，对于理解语境感知的系统，这两个概念都是基础性的。假定语境是“可用来表征一个实体的状况的任何信息”，见 Loke［LOK 06］的定义。位置、时间、计算资源和网络带宽是语境信息的例子。另外，假定状况意指一个给定实体当前状态的描述，也见 Loke［LOK 06］的定义。例如，具有低反应性、高报文丢失率和高时延（语境信息）的一条链路也许是拥塞的（状况）。在多数情形中，和在前面的例子中发生的情形一样，为确定一个实体的状况，不得不汇聚语境信息。

语境感知系统必须能够确定状况（如上所述，其中涉及一个实体）并检测现时状况的改变，结果是，感知到语境。语境感知也使系统能够自动化地执行动作，这运行在没有人类干预的条件下实施网络控制。为提供这些功能，首先，系统必须

监测环境，获取有关一个网元的语境信息。在此之后，必须就获得的语境进行推理，之后为取得一个目标，必须执行动作。各实体也必须与其他实体通信以便及时地确定它们的状况。为克服这项挑战，提出使用一个 MAS 范型作为一个建模基础 [SIL 07]。因为代理本身的性质，在开发一个自治引导系统方面，多代理范型似乎是有吸引力的。这些性质包括自治性、预测能力、自适应能力、协作和移动性。另外，多代理系统本质上是去中心化的，这是大型联网环境所需要的。但是，在这个项目中，通过评估一个特定问题，证明这样一个自治引导系统的概念。在将所提供解决方案扩展到大型问题之前，在一个有代表性的情形中对之进行试验是合理的，这是由于系统必须要处理的网络复杂性和大量（multitude）参数。目标是表明，将解决方案外推到整个网络是可能的，因为曾在一种比较简单的情形中评估了它的性能。

存在可被用来开发语境感知的几项技术。本章描述这些技术中的一些技术，并为开发引导系统给出选择。4.4.1 节描述语境感知系统的一个通用分层架构。下面各节基于这个架构的各层给出选择。首先，4.4.2 节重点讨论语境信息，这些信息可从物理网络和虚拟网络中获取。此后，4.4.3 节描述用来表示知识的几项技术。最后，4.4.4 节定义可被用来引导虚拟网络的动作。4.5 节给出结语。

4.4.1　语境感知系统架构

语境感知系统有三项基本功能：感知、思考和动作。遵循 Baldauf 等 [BAL 07] 提出的通用抽象分层架构，如图 4.8 所示。在这个通用架构中，这三项功能表示为子系统。为交换信息，每个子系统都由一层或相互关联的多层组成。但是，每个子系统可被解耦或紧密集成为一个设备，即一个子系统可思考和动作，但使用一个共享的传感器集获取语境信息。另外，这些功能中每项功能的复杂度等级都是独立的。例如，可开发这样一个系统，它有复杂的传感器，但在采取动作之前几乎不实施推理。此外，这三项基本子系统可以中心式或分布式方式加以实现。

感知是一个语境感知子系统，可被分成两层（见图 4.8）：传感器和原数据检索。第一层由一个传感器集合组成。假定传感器是每个数据源，它提供语境信息而不管数据源是“什么”。在这个意义上，温度的一个数据源可以是一个硬件设备（如一个温度计）或一个软件模块（如一项软件应用，它从一项 Web 服务请求温度）。这两者都被看作传感器，原因是它们都将温度读数提供给系统。感知子系统的第二层负责原语境数据的

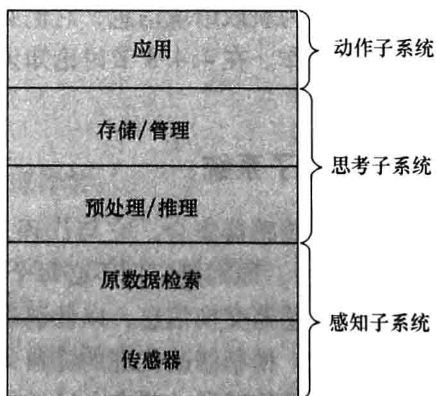


图 4.8　语境感知系统的一个分层架构模型

检索。该层向上层提供更抽象的方法，请求传感器获取的语境信息。实际上，该层是一个接口，使传感器的实现细节对思考子系统是透明的。因此，在上层中不做修改的条件下，可修改系统所使用的传感器。

在采用传感器收集数据之后，系统的任务是使用这样的数据，并使之具有意义 [LOK 06]。这是思考子系统的角色，可分为两层：预处理和存储/管理。预处理层将获取的所有语境信息转变为一个共同表示，因为语境信息可以不同形式获取，如离散值或值的一个连续系列。此外，该层进行推理并解释语境信息，以推断更多知识。系统使用的推理技术范围从简单的事件—条件规则到复杂的人工智能技术 [LOK 06]。预处理层也汇聚来自不同传感器的语境信息，以便提供更准确的信息。另外，需要知识表示技术，即以一种机器可处理的形式来定义和存储语境信息。这是存储/管理层的角色。存在几项知识表示技术，如基于图形的、基于逻辑的和基于本体的技术 [BAL 07]。

第三个子系统是动作子系统。基于采集的语境信息或由感知和思考子系统识别的状况，一个语境感知系统采取动作。依据应用需求定义由系统采取的动作，如图 4.8所示。例如，在一个拥塞控制应用中，传感器监测一条网络链路的带宽，并指明该链路是饱和的。因此，系统“思考”，并决定阻塞新流和维护所有的当前流。一般而言，一个语境感知系统必须及时动作，以便快速地依据环境变化调整其操作。另外，这些系统应该允许用户控制动作，即用户应该能够覆盖、取消和终止动作，且也可反转（reverse）一个动作的结果。

遵循所描述的架构来开发一个网络引导系统。基于 MAS，以一种分布式方式实现各层。感知子系统是采用沿虚拟路由器分散的传感器实现的。由此，不是采取每个代理一个传感器的做法，而是可由所有代理使用的一个传感器集合。通过直接读取存在于路由器操作系统上的数据和使用测量工具，传感器收集语境信息。也定义了传感器将获取的语境信息转换为一个共同表示。在 4.4.2 节中详细描述所考虑的语境信息和正在开发的传感器。各代理实现与思考和动作子系统有关的所有其他各层。因此，为获取语境信息，它们必须与传感器集合进行通信，并由此处理这个信息以采取动作。在 4.4.3 节讨论知识表示技术，在 4.4.4 节中讨论由引导系统定义的动作。

4.4.2 感知子系统

开发一个传感器集合，它与代理是解耦的。因此，能够开发轻量代理（以计算处理来衡量），原因是它们不必每个要被感知的语境信息都实现一个传感器。代理仅必须与传感器交换信息，以便请求和接收期望的语境信息。因此，虽然通过解耦感知子系统，稍稍修改代理的经典架构，但每个代理仍然能够获取感知信息。

正在开发的传感器，基本上通过使用两种方法获取语境信息。传感器读取由运行在物理路由器和虚拟路由器上的操作系统提供的可用数据，也使用著名的工具来

监测网络，如 ping [LIN 10]、nmap [NMA 10] 和 ifconfig [VAN 10]。在收集数据之后，传感器将原数据翻译为扩展标记语言（XML）[QUI 10] 数据结构，当被请求时，则已经准备好要发送到各代理了。本章将焦点放在语境信息的描述上，这些信息对于控制虚拟网络间的资源分配是有用的。为虚拟化一个网络，首先定义计划在多个虚拟网络间共享的资源。当前，识别出如下基本资源：处理能力、内存、带宽、流量和网络拓扑 [SHE 09]。网元（如路由器和交换机）的计算资源必须在虚拟网络间分片。这些资源包括处理能力和内存。值得指出的是，为使系统正常地工作，资源共享是一项基本要求。如果一台路由器没有可用的 CPU 周期，则它可停止转发报文和交换路由消息。当一台给定虚拟路由器的路由表增加时，为避免覆盖另一台虚拟路由器的一个路由表，内存隔离是一项重要要求，如果出现覆盖，则报文将不被正确地转发。带宽、流量和拓扑是必须分片的网络资源。首先，必须确保在一条物理链路上每个虚拟网络所占带宽比。在一条给定源—目的路径上能够限制转发速率的所有网元也都应该被分片 [SHE 09]。其次，必须能够将一个特定流量集与一个或多个虚拟网络关联，即流量集必须是相互隔离的。这是在多个虚拟网络中采用的一个关键点。就这个意义而言，流量意指如来去一个给定地址的所有报文或所有超文本传输协议（HTTP）流量。最后，每个网元应该知道一个给定虚拟网络内的各个节点。因此，一台虚拟路由器应该能够确定它自己对其他路由器以及这些路由器间的连通性的视图。

开发传感器，获取与基本资源有关的语境信息。因此，可从物理网络和虚拟网络获取的数据被分为两组：计算资源和网络状态。第一组与每台物理路由器和虚拟路由器有关。第二组指明每条物理链路和虚拟链路的状态。对于每台物理路由器，取决于所采用的网络虚拟化技术。传感器目前能够获取如下数据：

1）处理器使用情况（各物理 CPU）；

2）使用的内存；

3）可用的内存；

4）使用的交换内存；

5）可用的交换内存；

6）分配给一台给定虚拟路由器的总内存；

7）每台物理路由器的虚拟路由器数；

8）分配给一台给定虚拟路由器的虚拟处理器数；

9）为一台给定虚拟路由器定义的虚拟接口数。

各传感器直接从物理路由器的操作系统读取要求的大部分数据。通过对虚拟处理器的使用情况求和，估计物理处理器使用情况。

对于每台虚拟路由器，各传感器可获取如下数据：

1）处理器使用情况（各虚拟 CPU）；

2）使用的内存；

3）可用的内存；

4）使用的交换内存；

5）可用的交换内存。

依据所采用的虚拟化技术，通过使用由物理路由器的操作系统所提供的工具，得到虚拟 CPU 使用情况；直接从虚拟路由器的操作系统读取内存信息。就这个意义上而言，需要访问这些虚拟路由器。

第二组语境信息是与网络状态有关的。传感器目前能够获取每物理网络接口或虚拟网络接口的如下信息：

1）所接收的报文数；

2）接收到的字节数；

3）错误报文数；

4）在接收过程中丢弃报文数；

5）接收速率；

6）传输报文数；

7）传输字节数；

8）在传输过程中丢弃报文数；

9）传输速率。

各传感器也发现物理网络和虚拟网络的拓扑。因此，对于每个邻居，可确定如一条给定链路的可用带宽和延迟。在项目中，考虑以前给出的语境信息足以作为推理技术（正在开发的引导系统所采用的）的一个基础。采用这个信息，为保障每个虚拟网络的隔离和资源，引导系统将能够对环境的变化做出反应。

4.4.3 思考子系统

思考子系统的角色是使传感器获取的数据有意义，并使用语境信息对环境变化做出反应。基本上来说，思考子系统将知识表示技术与推理技术组合在一起。知识表示技术以一种机器可处理的形式定义和存储语境信息［DAV 93］。因此，一项知识表示技术的目标是以一种逻辑形式存储语境信息，以便使用这个信息支持推理技术。推理技术包括如数学模型、推断技术和基于认知的模型［LOK 06］。将焦点放在知识表示技术上。人们提出几项技术来表示知识，并存在将这些技术分为各类的研究［LOK 06，STR 04，BAL 07］。下面简短地描述了知识表示技术最有关类中的四个类。

基于标记的技术定义层次化数据结构来表示语境信息。这些数据结构由带有属性和内容的标记标签组成。每个标签的内容可递归地由其他标签定义。最流行的标记语言之一是 XML［QUI 10］。目前，存在几个基于 XML 的知识表示语言和标准，如 DARPA 代理标记语言（DAML）［DAR 10］和 Web 本体语言（OWL）［MCG 10］。当前正使用 XML 作为共同语言来表示由传感器获取的所有数据，也用之描述

在系统中所交换消息的内容。

一般而言，基于预定义的条件，一个逻辑从一个表达式集或事实推导得到一个结论性的表达式［STR 04］。这个过程被称作推断，且条件由一个规则集形式化地描述。结果是，如果正在考虑基于逻辑的技术，则语境信息可表示为事实、表达式和规则。因此，以事实的形式，语境信息被添加到一个基于逻辑的系统，删除或更新语境信息的情况类此。也可从系统中定义的规则推断语境信息。例如，Ranganathan 和 Campbell［RAN 03］建议通过使用一阶逻辑技术来表示语境和状况。在这种情形中，通过使用 Prolog 语言，定义规则将状况映射到动作。所提出的规则基本上将语境信息与状况相关。语境信息是一个规则的条件，状况是一条规则的结论。

由于其直觉特性，图形建模技术被大量使用。UML［OBJ 10］是一个著名的通用建模工具，并且也可用来表示语境信息［KOG 02］。UML 类图是这种语言的图形组件，并从这些图可推导实体—关系模型［BER 05］。这种模型大量用作开发关系数据库的结构化工具，这可被看作一个知识库［STR 04］。另一项图形技术被称作语境图［BRÉ 03，PAD 04］。这项技术没有像 UML 一样定义图（diagram），但它提出语境空间的概念，这是语境信息的一个空间视图。每种类型的语境信息表示为一个多维空间的一个轴，由此，传感器读数表示为点，状况表示为这个空间中的区域。

一般来说，本体被定义为一个概念和术语集合，被用来描述一个知识领域，或开发这个领域的一种表示法，这包括领域元素之间的关系。特别地，术语集可以一种层次化方式排序，并用作构建一个知识库的“草图”（sketch）［GÓM 99］。由这个定义，可清晰地识别本体和知识库之间的差异。本体提供一个术语集来描述一个给定的领域，而一个知识库用这些术语描述一个给定状况。如果这个状况改变，则知识库也改变。但是，因为领域保持一样的，所以本体没有改变。因此，可容易地从本体开发一个知识库，这是这项技术的主要优势。此外，基于本体的技术有其他优势。首先，这些技术避免不确定性，原因是它们提供一个准确描述和一个特定的词汇来表示知识。其次，基于本体的技术支持知识共享，原因是在同一领域内的各应用可使用同一本体。最后，同一本体可表示为不同语言。一般而言，一个本体由一个分类法（即一个概念集和概念之间的一个层次结构）、这些概念之间的一个关系集和一个公理集［GÓM 99］组成。

为引导系统构建一个知识库，由此定义一个本体来描述有多个虚拟网络的环境。另外，所分析的针对开发代理的多数平台都有工具来定义其代理内部的本体。Ginkgo 平台［GIN 08，GIN 09］考虑由类和个体组成的一种基于本体的表示法。这两个概念非常类似于一个面向对象数据模型的类和实例。一个个体是一个类的一个实例。因此，一个类是一个个体集（个体是这个类的成员），而知识库是一个类树，如图 4.9 所示（摘自参考文献［GIN 09］）。在这个例子中有两个类——Person

和 Employee，它们是从根类 Thing 派生的，每个类有一个实例——John 和 Jane。在实践中，类可以是路由器、用户、流、应用等。此外，可观察到类有存储数据的属性。这些属性有一个标识符，是单值的或多值的。个体有它们作为成员的类的相同属性，同样有其父类的属性。

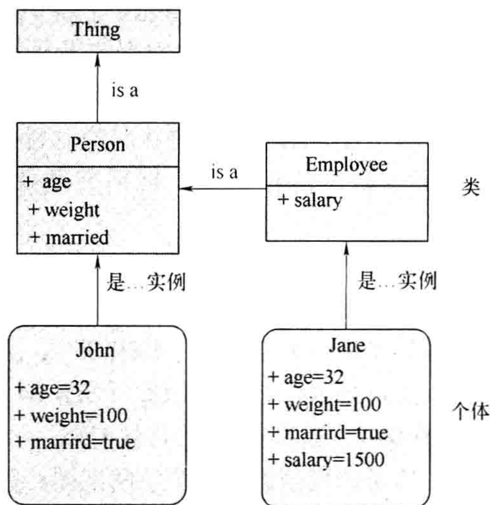


图 4.9 使用 Ginkgo 平台的一个简单知识库

4.4.4 动作子系统

动作子系统是引导系统的组成部分，负责基于采集的语境信息或由感知和思考子系统识别的状况，对环境的变化做出反应。我们的选择是，不管网络采用的虚拟化技术是什么，定义典型的基本功能。之后，基于这些功能，由引导系统采取动作。下面描述这些功能。

识别出四项基本功能：多个定制网络的创建、灵活的管理、实时控制和监测。为采用这些功能，也定义原语。各原语支持实例化/删除（instantiate/delete）、迁移（migrate）网元和流，以及设置（set）它们的资源分配参数。这种原语使网络虚拟化成为创建多个虚拟网络的一种合适技术，结果是支持多元论方法的合适技术，因为它满足了几项需求，如下解释。

1）在一个多元论架构中，有并行运行的多个定制网络，由此创建功能是基本的。就这个意义上来说，实例化原语可被用来实例化虚拟网元，如虚拟路由器和/或虚拟链路，且由此多个虚拟网络可被快速部署和同时运行。每个虚拟网络有其自己的协议栈、网络拓扑、管理策略等。这使网络创新和新的商务模型成为可能[FEA 07]。采用网络虚拟化，一个服务提供商可分配一条端到端虚拟路径，并针对所提供网络服务裁剪过的一个虚拟网络（如带有 QoS 保障的一个网络）实例化。因此，可容易地部署新服务，且新的玩家（players，局中人）可打破壁垒进入到网

络服务市场。

2）灵活的管理是识别出的第二项功能。网络虚拟化层打破了逻辑（用来构造转发表）和物理硬件（实现报文转发任务）［WAN 08］之间的耦合。因此，迁移原语支持将一个虚拟网元从一个物理硬件移动到另一个物理硬件，而不改变逻辑/虚拟网络拓扑。另外，流量工程和优化技术可使用迁移原语，沿物理基础设施移动虚拟网元/链路，目的是最小化能量成本、从服务器到特定网络用户的距离或其他目标函数。

3）实时控制是第三项功能。虚拟网络架构也支持虚拟网络资源的实时控制，因为可为每个虚拟网元（路由器、交换机、链路、网关等）设置资源分配参数。可设置分配的内存、带宽、最大容忍时延等，即使特定的硬件参数也可设置，如在一个竞争场景中虚拟处理器数和处理器使用的优先级。因此，依据当前网络状况、用户数、每个虚拟网络的优先级、服务水平协议（SLA）等，可动态地调整分配给每个虚拟网络的资源。

4）监测也是一项重要功能，原因是网络虚拟化技术要求一个监测工具集，这是测量关注的变量所需的，如可用带宽、处理器和内存使用以及链路和端到端时延。为测量期望的变量，调用监测原语。在这个项目中，由传感器实施测量，见4.4.2节的解释。由此，当调用监测原语时，实际上调用的是传感器。

引导系统使用的四项功能是保障每个虚拟网络的需求。例如，为检测恶意代码，一个给定虚拟网络采用一个入侵检测系统（IDS）。在这种情形中，如果监测到一次攻击［如分布式拒绝服务（DDoS）］，则可使用删除原语来删除一个虚拟网元/链路，甚至整个网络。图4.10也给出一个流量工程的例子，它使用迁移原语，沿物理基础设施移动虚拟路由器，以便最小化能量成本或其他目标函数。在这个例子中，有5个物理路由器（R_1、R_2、R_3、R_4 和 R_5），且最初有两个虚拟网络。第一个虚拟网络为IP上的话音（VoIP）呼叫提供QoS，并由分别放置于物理路由器 R_1、R_2 和 R_3 中的虚拟路由器 A_1、A_2 和 A_3 组成。第二个虚拟网络是一个安全网络，由分别放置于物理路由器 R_4、R_2 和 R_5 中的虚拟路由器 S_1、S_2 和 S_3 组成。假定一条负载均衡规则是刚好在物理CPU使用率达到80%之后，移动有最高CPU利用率的虚拟路由器。在这个场景中，如图4.10a所示，物理路由器 R_2 有两个虚拟路由器 A_2 和 S_2，且每个路由器使用物理CPU的25%。由此，满足所定义的规则。假定为视频流化应用提供QoS，创建第三个虚拟网络。如图4.10b所示，这个网络由分别放置于物理路由器 R_4、R_2 和 R_5 中的虚拟路由器 V_1、V_2 和 V_3 组成。因此，将一个虚拟路由器添加到 R_2，且 V_2 要求25%以上的物理CPU使用率。目前，CPU使用率是75%，且规则仍然得到满足。但是，在此之后，虚拟路由器 V_2 要求10%以上的CPU使用率，则总CPU使用率变得大于80%。因此，依据所定义的规则，必须移动具有最高CPU使用率的虚拟路由器，在这种情形中是 V_2。一种方案是将 V_2 移到 R_3，如图4.10c所示。在迁移虚拟路由器之后，所有物理路由器中的CPU

使用率小于 80%，并满足规则。

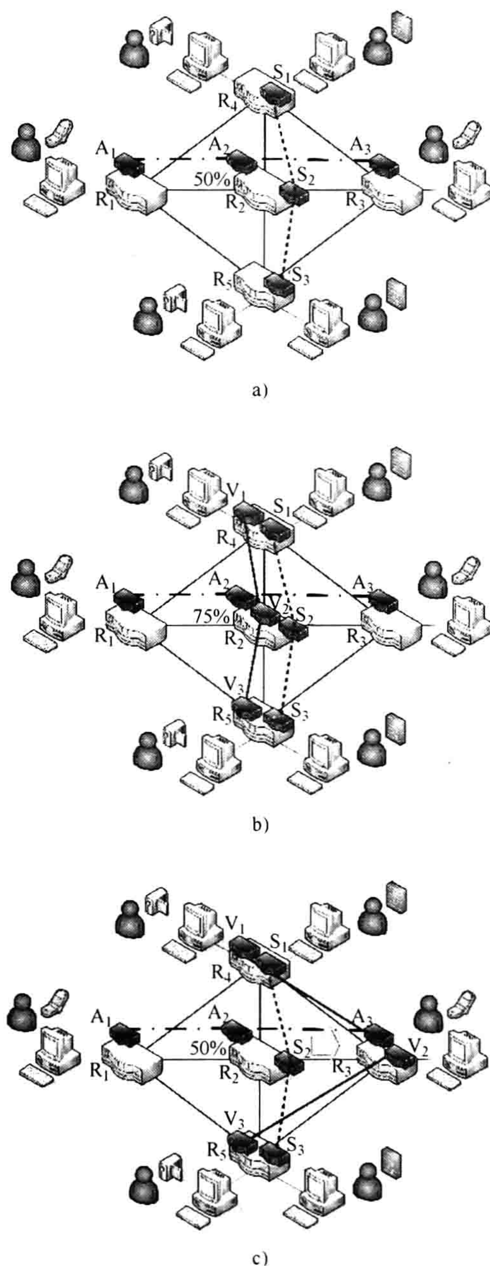


图 4.10 使用迁移原语的流量工程

a）在 R_2 中两个虚拟网络和 50% 的 CPU 使用率 b）在 R_2 中两个虚拟网络和 75% 的 CPU 使用率

c）虚拟路由器 V_2 迁移到物理路由器 R_3

4.5 小结

本章描述有关自治系统和 MAS 的最新技术状态。这些系统是良好地适应复杂环境的，并由实现自引导系统概念的高度动态的网络组成。给出构造代理的三个平台的概述，这对于网络控制和管理的开发是有用的。

开发一个引导系统所需的基础设施，该系统控制分配给每个虚拟网络的资源。这个系统创建和销毁虚拟网络，设置虚拟网络的参数并迁移网元。基于沿网元分散的传感器获取的语境信息，执行这些动作。引导系统是基于多代理范型的，为增加网络扩展性以一种分布式方式开发的。

引导系统也遵循 4.4.1 节讨论的分层架构。正在开发的传感器读取由运行于物理路由器和虚拟路由器上操作系统提供的可用数据，并且也使用监测网络的著名工具。但是，传感器解耦于代理，这使它们就计算处理方面是轻量的。在这种情形中，各代理仅必须与传感器交换信息，请求和接收期望的语境信息。目前，各传感器没有向代理发送消息。在收集数据之后，传感器将原数据翻译为要发送到一台服务器的 XML 数据结构。

为引导系统定义的动作是多个定制网络的创建、灵活的管理、实时控制和监测。另外，为利用这些功能而定义的原语是实例化、删除和迁移网元与流，并设置资源分配参数。不管网络采用的虚拟化技术是什么，这些功能和原语都是可行的。为评估这些功能和所定义的网络资源是否足以引导网络，则要求进行试验分析。

4.6 致谢

感谢 Carlos Roberto Senna、Daniel Macêdo Batista 和 Nelson Luis Saldanha da Fonseca，感谢他们在提升本章终稿质量方面所做的工作。

4.7 参考文献

[AND 05] ANDERSON T., PETERSON L., SHENKER S., *et al.*, "Overcoming the internet impasse through virtualization", *IEEE Computer*, vol. 38, pp. 34–41, April 2005.

[BAL 07] BALDAUF M., DUSTDAR S., ROSENBERG F., "A survey on context-aware systems", *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, pp. 263–277, June 2007.

[BEL 01] BELLIFEMINE F., POGGI A., RIMASSA G., "Developing multi-agent systems with JADE", *Intelligent Agents VII Agent Theories Architectures and Languages*, vol. 1986, pp. 89–103, 2001.

[BEL 03] BELLIFEMINE F., CAIRE G., POGGI A., *et al.*, "JADE a white paper", *Exp*, vol. 3, no. 3, pp. 6–19, September 2003. Available at http://jade.cselt.it/papers/2003/WhitePaperJADEEXP.pdf (accessed in May 2013).

[BER 05] BERARDI D., CALVANESE D., DE GIACOMO G., "Reasoning on UML class diagrams", *Artificial Intelligence*, vol. 168, nos. 1–2, pp. 70–118, 2005.

[BRÉ 03] BRÉZILLON P., "Representation of procedures and practices in contextual graphs", *The Knowledge Engineering Review*, vol. 18, pp. 147–174, June 2003.

[BRI 01] BRIOT J., DEMAZEAU Y. (eds), *Principes et Architecture des Systèmes Multi-Agents*, Hermes, 2001.

[BOR 05] BORDINI R.H., DASTANI M., DIX J., *et al.*, (eds), *Multi-Agent Programming Languages, Platforms and Applications*, Springer, 2005.

[BUL 08] BULLOT T., KHATOUN R., HUGUES L., *et al.*, "A situatedness-based knowledge plane for autonomic networking", *International Journal of Network Management*, vol. 18, pp. 171–193, March 2008.

[DAR 10] DARPA TEAM, *The DARPA Agent Markup Language Homepage*, June 2010. Available at http://www.daml.org (accessed in May 2013).

[DAV 93] DAVIS R., SHROBE H., SZOLOVITS P., "What is a knowledge representation?", *AI Magazine*, vol. 14, no. 1, pp. 17–33, 1993.

[DOB 06] DOBSON S., DENAZIS S., FERNÁNDEZ A., *et al.*, "A survey of autonomic communications", *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, pp. 223–259, December 2006.

[FAC 06] FACI N., GUESSOUM Z., MARIN O., "DimaX: a fault-tolerant multi-agent platform", *Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, SELMAS '06*, ACM, New York, NY, pp. 13–20, 2006.

[FEA 07] FEAMSTER N., GAO L., REXFORD J., "How to lease the Internet in your spare time", *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 61–64, January 2007.

[FER 95] FERBER J., *Les Systèmes Multi-Agents: Vers Une Intelligence Collective*, Dunod, 1995.

[FIP 10] FIPA, "The foundation for intelligent physical agents", October 2010. Available at http://www.fipa.org (accessed in May 2013).

[GIN 08] Ginkgo Networks, Ginkgo distributed network piloting system, Technical report, Ginkgo Networks, September 2008.

[GIN 08] Ginkgo Networks, "White paper–Ginkgo distributed network piloting system", September 2008. Available at http://www.ginkgo-networks.com/IMG/pdf/WP_Ginkgo_DNPS_v1_1.pdf (accessed in May 2013).

[GIN 09] Ginkgo Networks, Ginkgo agent platform programming manual v1.5, Technical report, Ginkgo Networks, October 2009.

[GÓM 99] GÓMEZ-PÉREZ A., "Ontological engineering: a state of the art", *Expert Update*, vol. 2, no. 3, pp. 33–43, 1999.

[HE 08] HE J., ZHANG-SHEN R., LI Y., *et al.*, "DaVinci: dynamically adaptive virtual networks for a customized internet", *Proceedings of the ACM CoNEXT Conference*, ACM, Madrid, Spain, December 2008.

[IBM 06] IBM, An architectural blueprint for autonomic computing, 4th ed., White Paper, June 2006.

[JES 08] JESS, "Jess, the rule engine for the java platform", November 2008. Available at

http://herzberg.ca.sandia.gov/jess/ (accessed in May 2013).

[KOG 02] KOGUT P., CRANEFIELD S., HART L., *et al.*, "UML for ontology development", *The Knowledge Engineering Review*, vol. 17, no. 1, pp. 61–64, 2002.

[LIN 10] Linux.org, *Ping Man page*, June 2010. Available at http://linux.die.net/man/8/ping (accessed in May 2013).

[LOK 06] LOKE S., *Context-Aware Pervasive Systems: Architectures for a New Breed of Applications*, 1st ed., Auerbach Publications, 2006.

[MCG 10] MCGUINNESS D.L., VAN HARMELEN F., *OWL Web Ontology Language Overview*, June 2010. Available at http://www.w3.org/TR/owlfeatures/ (accessed in May 2013).

[NMA 10] Nmap.org, *Nmap Reference Guide*, June 2010. Available at http://nmap.org/book/man.html (accessed in May 2013).

[OBJ 10] OBJECT MANAGEMENT GROUP, *UML Resource Page*, June 2010, Available at http://www.uml.org/ (accessed in May 2013).

[PAD 04] PADOVITZ A., LOKE S.W., ZASLAVSKY A.B., "Towards a theory of context spaces", *Proceedings of the Workshop on Context Modelling and Reasoning (COMOREA)*, pp. 38–42, IEEE, March 2004.

[QUI 10] QUIN L., *Extensible Markup Language (XML)*, June 2010. Available at http://www.w3.org/XML/ (accessed in May 2013).

[RAN 03] RANGANATHAN A., CAMPBELL R.H., "An infrastructure for contextawareness based on first order logic", *Personal and Ubiquitous Computing Journal*, vol. 7, pp. 353–364, December 2003.

[SHE 09] SHERWOOD R., GIBBY G., YAPY K.-K., *et al.*, FlowVisor: a network virtualization layer, Technical report, Deutsche Telekom Inc. R& D Lab, Stanford University, and Nicira Networks, October 2009.

[SIL 07] SILVA V., LUCENA C.J.F., "Modeling multi-agent system", *Communications of the ACM*, vol. 50, pp. 103–108, May 2007.

[STE 03] STERRITT R., BUSTARD D., "Towards an autonomic computing environment", *Proceedings of the 14th International Workshop on Database and Expert Systems Applications, DEXA '03*, IEEE Computer Society, p. 699, 2003.

[STR 04] STRANG T., LINNHOFF-POPIEN C., "A context modeling survey", *Proceedings of the International Workshop on Advanced Context Modelling, Reasoning and Management*, The 6th International Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, England, 2004.

[VAN 10] VAN KEMPEN F.N., COX A., BLUNDELL P., *et al.*, *Ifconfig Man page*, June 2010. Available at http://linux.die.net/man/8/ifconfig (accessed in May 2013).

[WAN 08] WANG Y., KELLER E., BISKEBORN B., *et al.*, "Virtual routers on the move: live router migration as a networkmanagement primitive", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 231–242, 2008.

[WOO 02] WOOLDRIDGE M., *An Introduction to Multi-Agent Systems*, John Wiley & Sons, 2002.

第5章　向虚拟网络提供隔离和服务质量

本章给出处理虚拟网络需求的一项提案，这些需求如隔离、服务质量（QoS）以及拥塞和故障避免。数据网络代表一个动态的和复杂的领域，其中管理人员每天都面临新问题和挑战。网络应用的快速增长和收集的日渐增多的信息量，使资源和网络控制变得越来越复杂。基础设施提供商和虚拟网络之间与共享相同物理基层的现有虚拟网络之间的冲突性目标，以及变化的需求，使虚拟网络的管理和控制成为一项困难的任务。因此，在这样一个不可预测的、变化的和开放的环境中，一种动态控制可以是取得改进的网络管理和监测的一种解决方案。之后提出自适应的网络监测方法，监测网络状态并编排网络的不同组件。主要目标是将一项智能控制添加到网络，来保障 QoS，同时改进网络管理和整体性能。

在本章，5.1 节描述现有控制算法，可在虚拟联网中加以使用。5.2 节描述使用 Xen 进行报文转发的主要挑战。之后，5.3 节给出针对物理网络节点内局部控制开发的一项提案。这项提案是在引导平面内设计的，是为确保来自所有虚拟网络的协议得到满足，即使存在行为不当违反其服务水平协议（SLA）的虚拟网络时也要得到满足。主要思想是控制域 0（dom0）共享的资源，依据虚拟网络的 SLA 调整每虚拟网络使用的资源量［FER 11］。最后，在 5.4 节给出小结。

5.1　虚拟网络控制和管理背景知识

可在局部范围和全局范围处理虚拟联网控制和管理。在全局范围，包括诸如实例化虚拟节点、虚拟链路和迁移虚拟网络等操作。另外，在局部范围，在每个物理节点上监测指派到每个虚拟网络的资源。例如，局部控制必须处理虚拟网络隔离。

Schaffrath 等提出虚拟网络全局控制的一个架构［SCH 09］。使用 Xen 实现的这项提案假定一个中心式控制，通过虚拟机和虚拟链路的实例化来创建网络分片。因此，在接收到分配一个新的虚拟网络的一条请求时，系统联系每个被选中的物理节点，并实例化构建整个网络所需的虚拟机以及使用互联网协议（IP）隧道法在这些虚拟机之间实例化虚拟链路。在基于虚拟化的测试床（如 GENI［GEN 08］）中发现了全局控制的其他类似方法。研究人员到测试床的访问是通过所谓情报交换所的一个中心实体加以控制的。情报交换所监测各联邦测试床的每个测试床上存在的物理节点和服务，它被授权使用它们，并确定为每名研究人员调度哪个分片［GUI 13］。

具有全局控制的实体也实施其他功能，如迁移［CAR 12］。Houid 等提出基于多代理的一个全局系统，使用迁移法动态地将资源分配给虚拟网络［HOU 10］。该

系统监测每个物理机器上的可用资源和虚拟网络的变化的需求。因为资源是稀有的，所以物理节点上的代理搜索一个类似的物理节点，接收它的一个或多个虚拟机。

全局网络控制系统不处理一个物理机器内的资源共享，相反它们假定各分片是由一个局部控制机制隔离的。Egi 等［EGI 07］深入考察了一个平台的构造，该平台是使用 Xen 和 Click［KOH 99］的虚拟路由器组成的。他们评估使用局部控制在网络间的隔离提供和公平性问题。作者深入考察使用不同数据平面的情况，假定路由通过一个特权域和虚拟机实施，并评估在虚拟网络间共享资源的能力。Egi 等扩展 Click 的 CPU 调度器，评估转发一条报文的 CPU 开销。但是，他们的工作没有定义为每个虚拟网络指派资源量的管理能力。同样，他们没有克服流量差异性以确保虚拟联网中的 QoS 问题。

局部控制的一个重要方面是保障虚拟化环境间的隔离。Xen 是有问题的，这不仅涉及输入/输出（I/O）操作，而且涉及其他方面，如公平性［HAN 09，JIN 09］。Jin 等提出一种机制，在 Xen 中确保 L2 和 L3 缓存利用率上的公平性，Xen hypervisor 的隔离机制是没有考虑这一点的［JIN 09］。所提供算法修改了 hypervisor 使用的内存页的分配法，使用页着色技术。

McIlroy 和 Sventek 基于 Xen 为当前互联网提出一种局部控制法。要求 QoS 的每条流被分配到一个虚拟机，称为 QoS routelet［MCI 06］。之后每个虚拟机将其 QoS 策略应用到到达流量间。该原型使用 Xen 实现，没有 QoS 需求的流量由特权域进行路由，而带有 QoS 的流量由虚拟机进行路由。作者指出，采用这个模型不能从最严格的意义上保障 QoS，原因是 Xen 调度器不适合这项任务。这项提案的另一个缺点是扩展性，原因是每条 QoS 流都需要一个虚拟机。

为其他虚拟化平台，人们也提出了隔离虚拟环境的各种机制。Trellis［BHA 08］是在虚拟网络基础设施（VINI）［VIN 10］上提供隔离的一个系统，VINI 是类似于 PlanetLab 的一个测试床。差异是，不像 PlanetLab，VINI 是作为其项目内的一个专用测试床发挥作用的。因为 VINI 是基于操作系统级上的虚拟化的，即所有虚拟环境共享相同内核，则相比采用平面隔离的 Xen 而言，使用 Trellis 的报文转发性能要低［EGI 08］。这些方法［BHA 08，WAN 08，ZEC 03］的主要问题是所有控制平面都执行在同一操作系统上，且缺乏为每个虚拟网络创建差异性数据平面的支持。

Genesis 是创建带有不同架构的虚拟网络的一个内核［KOU 01］。基于层次结构和继承的概念，Genesis 提出应该基于一个“root”（根）网络创建不同的“孩子”虚拟网络，孩子网络从根网络继承共同特点。类似于 Trellis，Genesis 基于这样的前提，即所有控制平面都工作在同一操作系统上。但是，对于每个虚拟网络，它允许使用不同策略和 QoS 机制。因为 Genesis 是在用户级实现的，并插入一个虚拟化层，所以它提供低的路由性能。

另一个虚拟化平台是 OpenFlow [MCK 08]，它基于简单转发单元组成的一个网络和一个中心式控制平面。为在虚拟网络间共享转发单元的物理资源，OpenFlow 提供 FlowVisor 工具 [SHE 10]。FlowVisor 可被看作转发单元和控制平面之间的一个透明代理。FlowVisor 在转发单元中控制 CPU 和内存的使用以及网络空间的分隔，即哪些特征定义每个虚拟网络。

Keller 和 Green [KEL 08] 提出基于 Linux 和 Click 创建一个共享数据平面的另一种方法。在这项提案中，每个虚拟网络可创建其自己的数据平面，基于通用假设，即在 Click 中进行报文转发。但是，作者没有讨论控制平面间资源的公平分割问题。

在本章中描述的提案将焦点放在局部控制和全局控制上。局部控制是由在 Xen 网络中每个物理节点内的隔离和 SLA 的工作实施的。全局控制自治地使用分布式容错算法解决物理节点失效，同时将寄居在过载物理节点上的虚拟网络迁移。

5.2 使用 Xen 进行报文转发中的挑战

使用 Xen 的虚拟网络模型认为，虚拟机就像路由器一样动作[○]。一个虚拟网络被定义为由虚拟路由器和链路组成的一个集合，是在物理基础设施之上创建的，如图 5.1 所示。Xen 架构是由一个 hypervisor、虚拟机 [称作用户域（domUs）] 和一个特权虚拟机（称作 dom0）组成的。Xen hypervisor 控制物理资源的访问，并处理由各个域实施的 I/O 操作。dom0 是直接访问硬件的一个特权域。因为 dom0 是一个驱动域，所以它存储所有物理设备驱动，并在放置于 domUs 的虚拟驱动和物理设备之间创建一个接口。另外，dom0 也是管理员和 hypervisor 之间的管理接口，创建虚拟机、改变 Xen 参数并管理 Xen 操作 [PIS 11]。

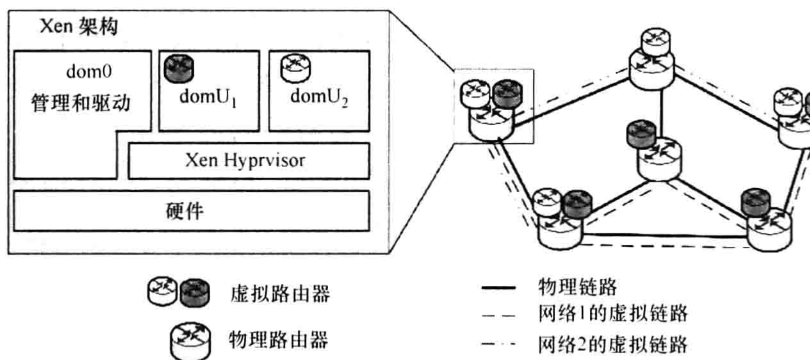


图 5.1 带有两个虚拟网络的 Xen 架构

○ 这项工作的一个以前的版本可在参考文献 [FER11] 中找到。

发送和接收报文是I/O操作，要求使用位于dom0处的设备驱动。因此，来自domUs的所有网络操作都产生dom0内存和CPU的一项额外负担。但是，Xen hypervisor没有高效地隔离dom0资源使用，这是Xen的一项主要弱点 [MAT 12]。表5.1表明，通过实施网络操作[○]，一个domU可容易地增加dom0 CPU消耗。因为在两个domUs之间和在一个domU和dom0之间的数据传递消耗dom0 CPU资源，所以在一个domU中一项恶意的或出现故障的动作，可容易地耗光dom0资源，这就损害了所有其他域的性能。因此，在虚拟化环境中管理和控制的目标是避免在一个虚拟网络上实施操作，这就影响其他虚拟网络，从而破坏了隔离前提条件。

表5.1 在dom0上的CPU消耗

CPU (%)	描 述
0.71 ±0.60	在dom0上的基本CPU消耗
66.43 ±8.93	从domU到dom0的TCP流量
85.49 ±5.91	从$domU_1$到$domU_2$的TCP流量
1.79 ±1.01	从一台外部机器到domU的TCP流量

对于网络操作，Xen常规架构不是高效的，原因是domU报文转发要走一条长的和慢速的路径。如图5.2a所示，报文到达dom0，之后到达domU，并返回到dom0，被转发到下一台路由器。平面隔离范型是改进转发性能的一种替代方法，原因是报文是由dom0中一个共享的数据平面直接转发的，如图5.2b所示。通过在dom0（可直接访问硬件）中维护domU的当前转发表的一个备份，完成平面隔离。重要的是指出，数据报文是直接由dom0转发的，但控制报文要被转发到domU，来更新控制平面。另外，平面隔离不会避免灵活的报文转发。如果一台虚拟路由器要求定制的操作（如监测或修改一个特定的首部字段），那么它们不能由共享的数据平面得到支持。在这种情形中，通过将一条默认路由插入到dom0内转发表的虚拟机中（就像常规报文转发中所做的那样），可忽略平面隔离。

最后，Xen也没有为QoS提供给出任何方案。由此，必须构建控制流量策略的一种方案，以便在一个虚拟网络内部和虚拟网络之间保障QoS。

○ 这些结果是在一台机器中采用top工具测量得到的，这台机器配有Intel Core 2 Quad处理器、4GB RAM和Xen 3.4-amd64。每个domU配置有一个虚拟CPU和128MB内存，且dom0配有一个虚拟CPU，没有内存约束。每个虚拟CPU与一个独占的物理CPU相关联。TCP流量是采用Iperf产生的。基本CPU消耗指在domUs中没有操作时dom0中的CPU使用情况。假定95%的一个置信区间。

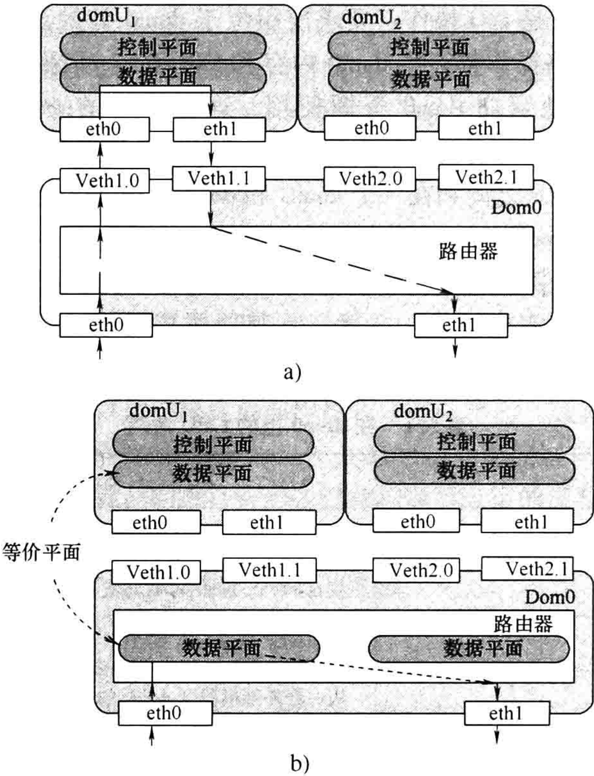


图 5.2 在基于 Xen 的网络中的报文转发

a）常规报文转发 b）使用平面隔离

5.3 控制域 0 共享的资源

域 0 资源（如 CPU、内存和带宽）是由 I/O 操作中的虚拟路由器共享的。因为在这种资源共享中没有严格控制，所以在报文转发过程中这些资源会被耗光，则就打破了虚拟路由器间的隔离。隔离失效会诱发安全问题和 SLA 违规。因此，为在每个物理节点中保障每个虚拟网络 SLA 而开发局部控制机制，就是必要的。所给出控制器的主要目标是向 Xen 虚拟化平台提供隔离，方法是控制 dom0 资源的使用。之后，依据物理机器的管理员所设置的参数，这个控制器分配和监测所有 domUs 使用的物理资源。

所给出的控制器称作最大使用控制器（MUC），为每个虚拟网络预留固定量的资源和称作权重的一个参数。这个固定的资源预留，保障一个最少量的资源可用于每个虚拟网络，而权重则指定在虚拟网络间如何分配空闲资源。只要存在空闲资源，它们就被提供给有需求的网络。实际上，这是选中用来控制物理资源使用率的策略。另外，这个控制器是基于资源监测和惩罚措施的。如果应用惩罚措施，则其严重程度依据的是每个虚拟网络 SLA 值的违反程度。同样，它支持平面隔离范型。因此，独立于数据平面位置，各个资源被正确地加以监测。

下面分析假定所有 domUs 都是不可信的，因为每个 domU 都有一个不同的管理员。因此，一个 domU 或有意或无意地伤害其他域。在所提供模型中，一个 domU 是恶意的或呈现一个不可接受的行为。当一个 domU 有意地执行破坏虚拟网络间隔离的一个动作并之后损害其他 domUs 的性能时，就发生了一个恶意行为。另外，一个不可接受的行为，是来自一个 domU 的超过保留给它的资源量的任何尝试，它试图使用所有的 dom0 空闲资源。这个行为可能干扰其他域的操作。因为这两种行为都是有害的，所以执行任何一种行为的虚拟机均被分类为敌对域，而其他域被分类为普通域。

下面介绍最大使用控制器。

MUC ［FER 11］分配并检测每 T 秒 dom0 资源的总体使用情况 $U(t)$ 和每个虚拟路由器 i 的相应使用情况 $U_i(t)$。dom0 资源可采用固定预留或应需的方式进行分配。基于固定预留的分配是由管理员处理的，它为每个 domU 预留一个固定量的 dom0 资源，为每个虚拟网络确保最小的质量。相反，应需分配目标在于资源使用率方面的高效率，因为 MUC 将空闲资源重新分布在 domUs 间，其需求大于预留的固定量。除了没有在使用的预留资源外，空闲资源还由非预留资源组成。因此，控制器的一个前提条件是只要存在一个需求，就提供一个虚拟路由器 i 的固定资源，表示为总 dom0 资源 $R(t)$ 的一个百分比 α_i。另一个前提条件是依据以前由管理员指派的优先级，应需地将所有空闲资源分配给虚拟路由器。这个优先级称作权重，表示为 W_i，其中 $\{W_i \in \mathbf{Z} \mid 1 \leqslant W_i \leqslant 1000\}$。一个虚拟路由器的权重越高，则它在 dom0 上具有访问权限的空闲资源就越多。因此，应需分配为每个虚拟网络提供一项附加的区分性服务。

通过观测在每个输出物理链路上传输的比特体量，MUC 监测带宽。如果一台路由器在一条输出链路中超过分配的带宽，它就受到惩罚，丢弃它的报文（在那条链路上发送的）。

基于通过 dom0 传递的报文体量，监测 dom0 中的 CPU 使用率。之后，被监测的数据以每项网络操作的开销加权。依据报文源和目的地，指派报文处理开销，因为如表 5.1 所示，一条报文对 dom0 CPU 利用率的影响取决于报文是来自一个 domU 还是去往 domU，或进出一台外部机器。如果一台路由器超过被分配的 CPU，则它受到惩罚。因此，为避免产生不公平 CPU 惩罚的攻击，重要的是为每项测量操作定义负责的域。在 domUs 之间传递时，发送报文的 domU 负责所有的 CPU 开销，以避免一个敌对域发起非请求流量耗光一个共同域的 CPU 资源。此外，在 domU 和 dom0 之间传递时，使用 CPU 的开销总是由 domU 负责。

通过观测每台虚拟路由器的转发表尺寸，MUC 控制内存使用率。如果 dom0 的内存达到临界限，则使用表或过滤器的所有虚拟路由器，如果占用的内存大于固定预留量，则受到惩罚，方法是通过丢掉（disposal）一定百分比的路由。为避免报文丢失，添加一条到虚拟路由器的默认路由。匹配一条被丢弃路由的各条报文，由虚拟路由器转发，而不是由 dom0 丢弃。结果，降低路由表尺寸不会诱发报文丢弃，但仅导致降低的转发性能，原因是报文由 domU 转发，而不是由 dom0 转发。

1. 惩罚计算

通过丢弃它们的报文或路由，对敌对域实施惩罚。MUC 搜索和收敛到一个丢弃概率，依据固定预留和权重值，由之平衡虚拟路由器间 dom0 资源的使用。为避免当在物理机器上还有空闲资源时的丢弃情况出现，一台虚拟路由器受到惩罚，仅当其使用率超过其固定的预留值并且总资源利用率达到一个临界水平，这由 dom0 中总资源 $R(t)$ 的一个百分比 β 给定。在没有空闲资源的条件下，使用的资源大于其固定预留值的所有节点都要受到惩罚，以避免其他虚拟路由器受到伤害。假定总的非预留资源由 $D(t)=R(t)-\sum_{\forall i}\alpha_i R(t)$ 给定，那么在 $t+T$ 中的丢弃概率由 $\Phi_i(t+T)$ 给定，依据下列算法 5.1 进行更新。

算法 5.1　惩罚计算的启发式方法

```
   input  : Φ_i(t), W_i, α_i, R(t), U(t), U_i(t), D(t), β
   output: Φ_i(t + T)
1  if (α_i · R(t) < U_i(t)) or (Φ_i(t) > 0) then
2  |  if (α_i · R(t) < U_i(t)) then
3  |  |  % Calculate an idle resource usage indicator
4  |  |  ϒ_i(t) = (U_i(t) − α_i · R(t))/D(t)
5  |  |  if (β · R(t) ≤ U(t)) then
6  |  |  |  % Since there are no idle resources, some networks can be
   |  |  |  damaged. Thus, the punishment is increased.
7  |  |  |  if (Φ_i(t) > 0) then
8  |  |  |  |  Φ_i(t+T) = min(Φ_i(t)+(1+ϒ_i(t))·(1+1/W_i)·Φ_i(t)/(3−1/W_i), 1)
9  |  |  |  else
10 |  |  |  |  Φ_i(t + T) = Φ_initial  % Set initial punishment
11 |  |  |  end
12 |  |  else
13 |  |  |  % Reduce punishment, because there are idle resources
14 |  |  |  Φ_i(t + T) =
   |  |  |  max(Φ_i(t) − (1 + (1 − ϒ_i(t))) · (1 − 1/W_i) · Φ_i(t)/(3+1/W_i), 0)
15 |  |  end
16 |  else
17 |  |  % Reduce punishment, because the router used only its fixed
   |  |  resources.
18 |  |  Φ_i(t + T) = max(Φ_i(t) − 3 · (1 − 1/W_i) · Φ_i(t)/3, 0)
19 |  end
20 else
21 |  Φ_i(t + T) = 0
22 end
```

重要的是即使一个 domU 比其固定预留值消耗较少的资源，为避免不稳定性，惩罚也不会立刻重置。同样，为防止由于 CPU 过载而由虚拟路由器产生的流量中断其他 dom0 服务，一个由滞后效应的惩罚总是恒定地应用在虚拟机的输出接口。这样的惩罚应该足够小，从而不会影响低体量传输，但应该防止一个 domU 消耗掉所有的 dom0 资源。

2. MUC 原型描述和分析

在存在敌对域的情况下，通过分析 MUC 的有效性来验证控制器共享资源的效率。采用 C 和 Python 实现了一个原型，且控制器能够监测 dom0 的带宽和 CPU。监测和惩罚是采用 Iptables 实现的。为动态地估计物理链路的容量，使用 Mii-tool。采用 MUC 监测 CPU 利用率，方法是估计每项网络操作在 dom0 中的开销，这包括 domUs 之间（从 domU 到 dom0 以及相反方向，从 domU 到一台外部机器以及相反方向）和外部机器之间的通信。输出虚拟接口中的滞后效应惩罚值估计为 0.0009，这依据的是报文速率，严重地影响 dom0 的响应时间。

测试是运行在一台机器上的，此后称为路由器，装备有一个 Intel Core2 Quad 处理器，有 4GB RAM，5 个物理千兆以太网接口，使用 Xen 3.4-amd64 路由器模式。实例化的所有 4 台虚拟机都运行 Linux 内核 2.6 26-2 的 Debian 操作系统，每个虚拟机都有一个虚拟 CPU、128MB 内存和 5 个网络接口。依据测试不同，dom0 中虚拟 CPU 数是变化的，且对这个域没有内存约束。物理 CPU 是在所有虚拟 CPU 间共享的，且 hypervisor 动态地将虚拟 CPU 映射到真实 CPU。测试使用两台外部机器产生和接收报文，每台机器有一个 1Gbit/s 的网络接口。所有流量是采用 Iperf 产生的，且结果有 95% 的置信区间。

第一项测试评估安全数据平面更新的可用性。如果 domU 安全地更新了其数据平面，且没有来自其他域的操作可防止其完成，则这项测试被认为是成功的。因此，分析了在数据平面更新过程中 MUC 利用率的影响。测试由从 $domU_1$ 更新数据平面的最多三次尝试组成，而 $domU_2$ 向 $domU_1$ 发送 TCP 流量。该场景仿真一个敌对虚拟路由器 $domU_2$ 尝试防止一个正常路由器 $domU_1$ 正常地操作。在 MUC 中，尝试更新数据平面的 $domU_1$ 有 $\alpha_1 = 0.5$，而敌对方 $domU_2$ 有 $\alpha_2 = 0.5$。所有 domUs 都有权重 $W = 500$。

图 5.3a 和图 5.3b 分别给出一个数据平面更新的成功概率和两台虚拟机之间传输的数据体量。当使用常规平面隔离时，即使在 dom0 中存在大量 CPU，来自 $domU_2$ 的一次攻击也是有效的。但是，MUC 增加了一次成功数据平面更新的概率，高达 100%。实际上，MUC 限制来自 $domU_2$ 的攻击流量，避免了 dom0 资源的过载。同样，MUC 预留 $domU_1$ 要求的 CPU 资源来发送更新消息并实施安全平面隔离所需的密码学操作。图 5.3b 给出从 $domU_2$ 到 $domU_1$ 的 MUC 惩罚流量，确保 dom0 CPU 资源不被耗光。因此，当使用 MUC 时取得的吞吐量，就小于当在 dom0 中仅有一个 CPU 时使用安全数据平面更新的情况。当 dom0 中的 CPU 数

增加时，CPU 约束条件就得到放松，那么使用 MUC 的吞吐量就增加。实际上，MUC 吞吐量甚至大于安全数据平面隔离的吞吐量。MUC 确保来自 $domU_1$ 的固定资源，那么 $domU_1$ 可处理数据平面更新和由 $domU_2$ 启动的 TCP 连接的 ACK 消息。对于吞吐量而言，丢失 ACK 消息的影响要比 MUC 对流量施加的限制要差，这是因为 CPU 消耗导致的。因此，作为带有控制器模块的所提供架构的一个结果，MUC 确保以高可用性实施一次安全的数据平面更新，同时确保虚拟机之间的一条高性能连接。

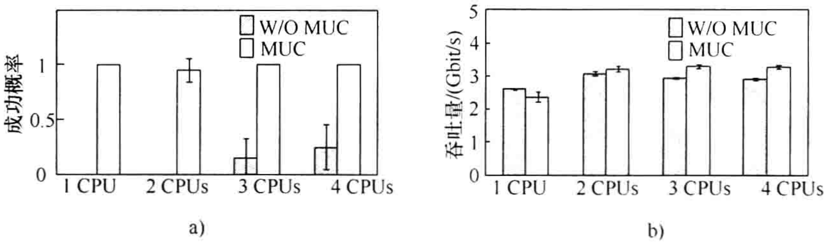


图 5.3　当使用 MUC 时安全数据平面更新的可用性

a）成功数据平面的概率　b）$domU_1$ 和 $domU_2$ 之间的吞吐量

第二项测试评估使用带有平面隔离的 MUC 时的传输时延。相比于没有任何控制的平面隔离范型的使用情况，评估了 MUC 的影响。这项测试测量依据 dom0 工作负载由 MUC 额外负担导致的时延。因为在资源共享中没有评估公平性，以 100% 的一个固定预留值，创建一个虚拟网络。该测试由两个试验组成，它使用 Ping 测量两台外部机器之间的往返时间（RTT）。在第一个试验中，没有背景流量，而在第二个试验中，在这两台外部机器之间产生背景 TCP 流量。在图 5.4a 中给出两个试验的结果。在没有背景流量情况下，数据传输给出两种配置的低 RTT。不过，当存在背景流量时，dom0 CPU 是过载的，这就增加了系统的响应时间，结果就增加了 RTT。结果表明，由 MUC 提供的 CPU 和带宽控制，防止 dom0 CPU 变得过载。由此，MUC 给出比常规平面隔离配置小 8 倍的一个 RTT。重要的是指出，即使 MUC 控制会诱发丢弃报文，这些丢弃不会导致对流量的大影响，如图 5.4b 所示。

第三个试验涉及共享输出链路。为做到这一点，在不同虚拟网络上的一个 domU 和一台外部机器发起与另一台外部机器的通信。因此，两个网络共享到目的机器的输出链路。两个网络均等地访问物理资源，在两台虚拟路由器中 $\alpha=0.5$ 和 $W=500$。为评估 MUC，使用层次结构令牌桶（HTB），也测试了使用流量控制（TC）的带宽控制。HTB 创建两个输出队列，每个队列有 512Mbit/s 的最小带宽和高达 1Gbit/s 的最大带宽，来仿真与 MUC 相同的资源使用策略。当 domU 以 1.5Gbit/s 的最大速率发送用户数据报协议（UDP）流量和 1500B 的报文，而外部机器发送 TCP 流量时，图 5.5a 和图 5.5b 给出所得到的结果。在开始，在网络中没

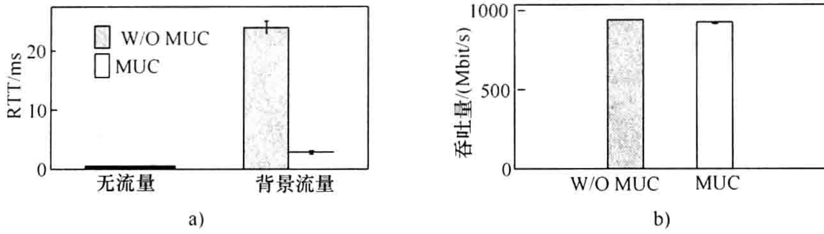


图5.4　MUC对RTT的影响

a）MUC对在dom0上有一个CPU的两台外部机器之间RTT的影响

b）在dom0上有一个CPU的两台外部机器之间的背景流量吞吐量

有控制。

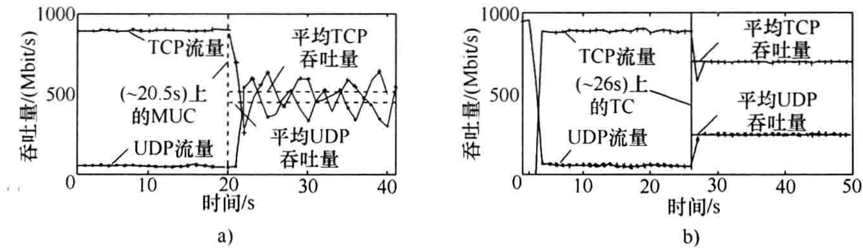


图5.5　使用MUC的资源共享控制

a）具有来自domU的UDP流量和来自外部机器的TCP流量的MUC

b）具有来自domU的UDP流量和来自外部机器的TCP流量的TC

结果表明，当没有资源共享控制时，外部机器流量的优先级高于domU流量。当使用平面隔离时，这是一次隔离失效，原因是外部流量影响由一个domU产生流量的最大体量。因此，属于一个敌对网络的一台外部机器，可注入流量损害另一个虚拟网络的一台虚拟路由器的性能。在这个试验中，可用资源是均等地在这两个虚拟网络之间共享的。那么，两个网络应该有一个相等的链路分片，这意味着每个虚拟网络有512Mbit/s。虽然MUC引入比TC大得多的流量方差，但就每个虚拟网络512Mbit/s的理想速率而言，MUC平均吞吐量比TC平均吞吐量有一个较小的误差（error）。事实上，如果对外部机器进入流（inflow）没有控制，则来自虚拟机的UDP流量是没有特权的（underprivileged），且不能取得高速率。因此，就512Mbit/s的理想速率而言，对UDP流量，MUC控制给出-14.2%的最大吞吐量误差，对于TCP流量给出-0.62%的最大误差。另外，就512Mbit/s的理想速率而言，对UDP流量，TC控制给出-52.18%的最大吞吐量误差，对于TCP流量给出+35.68%的最大误差。这个结果表明，在链路资源共享方面，MUC给出较高的公平性，原因是它适应到了Xen架构特性。

5.4 小结

本章解释了控制虚拟网络 SLA 的一种开发的算法。这项提案的内部操作及其接口都详细提供给引导平面，用来依据期望的策略和原语，控制虚拟网络环境。此外，结果表明，对指派给每台虚拟路由器的资源编排到每个虚拟网络是可能的。

5.5 参考文献

[BHA 08] BHATIA S., MOTIWALA M., MUHLBAUER W., *et al.*, Hosting virtual networks on commodity hardware, Technical report, Princeton University, Georgia Tech, and T-Labs/TU Berlim, January 2008.

[CAR 12] CARVALHO H.E.T., DUARTE O.C.M.B., "Elastic allocation and automatic migration scheme for virtual machines", in *Journal of Emerging Technologies in Web Intelligence (JETWI)*, Academy Publisher, no. 4 vol. 4, November 2012.

[EGI 07] EGI N., GREENHALGH A., HANDLEY M., *et al.*, "Evaluating Xen for router virtualization", *International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, Hawaii, USA, pp. 1256–1261, August 2007.

[EGI 08] EGI N., GREENHALGH A., HANDLEY M., *et al.*, "Fairness issues in software virtual routers", *ACM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Seattle, WA, USA, pp. 33–38, August 2008.

[FER 11] FERNANDES N.C., DUARTE O.C.M.B., "XNetMon: a network monitor for securing virtual networks", *IEEE International Conference on Communications (ICC 2011 - Next Generation Networking and Internet Symposium (NGNI))*, Kyoto, Japan, pp. 1–5, June 2011.

[GEN 08] GENI PROJECT OFFICE, "Geni spiral 1 overview", September 2008. Available at http://groups.geni.net/geni/attachment/wiki/SpiralOne/GENIS1Ovrvw092908.pdf.

[GUI 13] GUIMARÃES P.H.V., FERRAZ L.H.G., TORRES J.V., *et al.*, "Experimenting content-centric networks in the future internet testbed environment", in *Workshop of Cloud Convergence: Challenges for Future Infrastructures and Services (WCC-02) - ICC'2013*, Budapest, Hungary, June 2013.

[HAN 09] HAN S.-M., HASSAN M.M., YOON C.-W., *et al.*, "Efficient service recommendation system for cloud computing market", *2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS'09)*, Seoul, Korea, pp. 839–845, November 2009.

[HOU 10] HOUIDI I., LOUATI W., ZEGHLACHE D., *et al.*, "Adaptive virtual network provisioning", *Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA 10)*, New Delhi, India, pp. 41–48, September 2010.

[JIN 09] JIN X., CHEN H., WANG X., *et al.*, "A simple cache partitioning approach in a virtualized environment", *IEEE International Symposium on Parallel and Distributed Processing with Applications*, Chengdu, China, pp. 519–524, August 2009.

[KEL 08] KELLER E., GREEN E., "Virtualizing the data plane through source code merging",

ACM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO), Seattle, WA, USA, pp. 9–14, August 2008.

[KOH 99] KOHLER E., MORRIS R., CHEN B., *et al.*, "The click modular router", *Operating Systems Review*, vol. 5, pp. 217–231, December 1999.

[KOU 01] KOUNAVIS M.E., CAMPBELL A.T., CHOU S., *et al.*, "The Genesis kernel: a programming system for spawning network architectures", *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 19, pp. 511–526, March 2001.

[MAT 12] MATTOS D.M.F., FERRAZ L.H.G., COSTA L.H.M.K., *et al.*, "Virtual network performance evaluation for future Internet architectures", in *Journal of Emerging Technologies in Web Intelligence (JETWI)*, Academy Publisher, no. 4 vol. 4, November 2012.

[MCI 06] MCILORY R., SVENTEK J., "Resource virtualisation of network routers", *IEEE Workshop on High Performance Switching and Routing (HPSR)*, Poznan, Poland, pp. 15–20, June 2006.

[MCK 08] MCKEOWN N., ANDERSON T., BALAKRISHNAN H., *et al.*, "OpenFlow: enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, April 2008.

[PIS 11] PISA P.S., COUTO R.S., CARVALHO H.E.T., *et al.*, "VNEXT: Virtual NEtwork management for Xen-based Testbeds", *2nd IFIP International Conference Network of the Future (NoF'2011)*, Paris, France, November 2011.

[SCH 09] SCHAFFRATH G., WERLE C., PAPADIMITRIOU P., *et al.*, "Network virtualization architecture: proposal and initial prototype", *ACM Workshop on Virtualized Infrastructure Systems and Architectures (VISA)*, Barcelona, Spain, pp. 63–72, August 2009.

[SHE 10] SHERWOOD R., CHAN M., COVINGTON A., *et al.*, "Carving research slices out of your production networks with OpenFlow", *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 129–130, January 2010.

[VIN 10] BAVIER A., FEAMSTER N., *et al.*, "In VINI veritas: realistic and controlled network experimentation", *ACM SIGCOMM Computer Communication Review*, New York, NY, USA, vol. 36, no. 4, pp. 3–14, August 2006.

[WAN 08] WANG Y., KELLER E., BISKEBORN B., *et al.*, "Virtual routers on the move: live router migration as a network-management primitive", *ACM SIGCOMM*, Seattle, WA, USA, pp. 231–242, August 2008.

[ZEC 03] ZEC M., "Implementing a clonable network stack in the FreeBSD kernel", *Usenix Annual Technical Conference (USENIX 2003)*, Santo Antonio, Texas, USA, pp. 137–150, June 2003.

第6章　引导系统

人们提出自治联网［DOB 06，GAÏ06，CHE 06］来处理通信网络的日渐增加的复杂性，且它是自治计算中的一个特定专题［KEP 03］，是由 IBM 杜撰的一个术语，通过支持系统的自管理来处理它们的复杂性。目的是将处理要求人类干预的任务（如设置管理策略）和促进任务的自动化（如系统配置和优化、灾难恢复和安全）的网络管理人员解放出来。

自管理也是鼓吹未来互联网应该采用架构的多元论方法的一个核心要素［TUR 05］。这种多元论方法表明，网络提供商应该被分成服务生产商和基础设施提供商［FEA 07］，且为做到这一点应该采用虚拟化［AND 05］。在这些网络中，用户向服务提供商请求网络服务，后者在网络基层上实例化虚拟网络（VN），网络基层由基础设施提供商提供。每个 VN 可有其自己的协议，确保运行于其上之服务目标的实现。此外，VN 的隔离是一项需求，即一个 VN 的操作运行不应该干扰其他 VN 的那些操作运行。由于满足这些需求所需的复杂性，为将资源以一种最优的、鲁棒的和安全的方式分配给各 VN，是非常有挑战性的［ZHU 06，YU 08，HOU 08］。

Horizon 项目目标是定义和确认基于多元论和知识平面（KP）［CLA 03］的一个新网络架构，具有基于高级指令、自动检测和问题纠正的自我重新配置能力。为取得这样的目标，就有必要有一个引导平面（PP）负责决策过程。本章报告通过为 Horizon 项目中提出的 PP 使用一个自管理系统得到的结果，目的是决定使用控制和管理实体，这些实体将网络和服务的语境考虑在内。

目标是在 Horizon 项目的设计中采用 PP，通过使用一个多代理系统而处理 VN。Horizon 架构中的引导方法是基于引导代理（PA）的，它们可以联邦方式工作。

本章组织如下。6.1 节在自治引导系统（APS）中引入一些概念。6.2 节讨论 PP 功能和需求。6.3 节给出一个初步的 PP 设计，而 6.4 节给出引导代理架构。6.5 节介绍用来验证自管理系统的测试床。在本节也给出用于第一个测试床的工具和一些初步试验。6.6 节给出这个初步的自管理系统的描述和多代理模型。6.7 节报告最终试验的结果。6.8 节给出第二个自管理器型多代理系统，但是，焦点放在 VN 的自愈方面。在这个语境中实施了一组试验。最后，6.9 节给出小结。

6.1　自治引导系统

APS，正如最初由 IBM 宣言中所定义的［IBM 06］，已经为单个系统的管理系统所采用，且假定它们可实施涵盖各种节点、链路和服务的不同管理任务。不同标

准的存在，使设计单个自治控制环是不现实的，这样的控制环自动地调整不同操作运行方面，如故障、配置、计费、性能和安全（FCAPS）。这意味着存在为每个方法定义一个或多个自治环的需求，从而使每个控制环的设计可以被简化。此外，网络的操作运行是所有控制环交互作用的结果，但它们也许有冲突的目标。例如，使用一个重型加密方案提高安全性的一个自治安全组件，可能要求太多的计算和带宽，将最大网络吞吐量限制到服务水平协议（SLA）建立的性能水平之下。另外，如果两个子网络具有冲突的配置，或者需要一个重新协商和重新配置过程，或者在这两个子网络的边界中必须安装转换服务（网关）。

为解决这样的一个问题，提出一个新的平面 PP，支持各种自治控制环的协作，确保各决策不会冲突。这种协作（引导）使自治组件和控制协议之目标的整体优化，与为整个网络定义的目标和 SLA 是一致的。引导也意味着不同运营商采用的自治管理域能够自动地调整它们的配置，来处理各网络的联邦（关系）。实际上，对一个 PP 的需求，来自具有不同管理目标的几个自治控制环的开发，它们自己是不能互操作的，从而支持协商、联邦和部署功能。因此，引导处理各 APS 的元管理，即自治管理控制环的部署和重新配置，以便支持它们的互操作。这是在一个高级目标集的基础上取得的，这些目标是为组成该网络的被管网络域定义的。新 SLA 和策略的协商，在被引导网络内，冲突的管理系统的去活（de-activation），后跟其他系统的激活和这些系统的迁移，可能需要方便网络的管理能力。整个引导过程遵循引导策略，策略规定的折中方案是每个管理域应该针对互操作性做些工作。

6.1.1 架构

VN 环境的架构可由四个平面组成：

1）转发数据的数据平面；

2）包含所有控制算法的控制平面；

3）管理平面（MP）负责所有管理功能；

4）PP 实时地将信息反馈到管理和控制平面。

数据平面负责将数据从一个发送者发送到一个或几个接收者，这可在利用虚拟化的一个网络中虚拟化单元。管理和控制平面为故障检测、诊断、安全管理、路由、流控、安全控制、移动性等提供算法。实际上，在各节点处可以有用于同一任务的不同算法。特定算法可能取决于网络和服务的语境与需要。

图 6.1 形象地给出这种自治架构的一个通用视图，其中给出所引入的 PP，汇聚两个特定子平面：KP 和编排平面（OP）。KP 应该能够快速地恢复知识（对于馈入控制和管理算法是有用的），且 OP 应该协调（同步）网元（NE）。在实现中这两个平面不能解关联，因此它们需要集成在一起。此外，OP 需要有引导智能过程的知识。这个 PP 的首次实现是作为一个元控制平面提出的 [GAÏ96]。[GAÏ96] 和 [BUL 08a] 详细描述这个架构的一些部分，[BUL 08b] 提出一些例子。与经典

架构有关的另一项所提出改变，是将管理和控制平面融合到恰好一个平面中，得到一个三平面自治架构（见图6.2）。

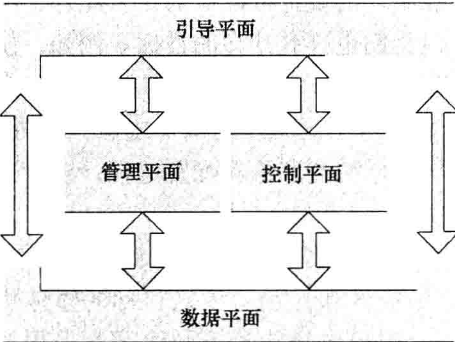


图6.1　自治架构的一个通用视图

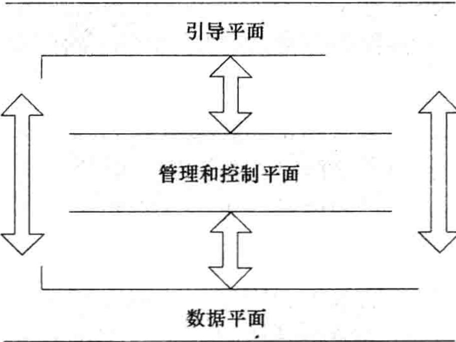


图6.2　面向引导的架构

PP使用控制算法驱动网络。出于这个目的，PP必须将信息提供给管理和控制算法。总之，PP必须编排管理和控制平面，它们配置数据平面。为做到这一点，一个分布式智能代理系统支持完成一个自适应控制过程，原因是每个代理持有不同过程［行为和动态规划器（DP）］，这可支持及时的和最相关的决策。各代理是以如下意义上隐性地协作的，即考虑到邻居的状态，它们使用一个共置视图。

6.1.2　Horizon项目的引导平面

作为对变化语境的响应和符合高级目标与策略，PP的目标是监管和集成网络控制的行为。它管理和集成所有其他平面的行为，这确保了管理操作的完整性。PP可被看作一个控制框架，为取得所要求的功能，任意数量的组件可被插入其中。此外，PP可与其他PP建立联邦。PP也优化网络监测，并确保当要求所需的知识时，它们是可用的。为管理长期过程，PP可使用局部可用的知识或全局知识。PP寄居几个APS，且它涉及一个或多个PA，一个动态知识库（KB）由一个数据模型和本体集以及合适的映射逻辑组成。每个APS代表一个虚拟实体集，使用一个共同的策略和知识集，虚拟实体管理一个虚拟设备或网络集合。各APS访问一个KB，KB由一个数据模型和本体集组成。可使用各行为，各APS相互通信和协作，各行为作为APS通信的桩（stub）。PP作为所有APS的控制工作流，确保启动、初始化、动态重新配置、适配和语境化以及优化。PP支持服务生命周期管理，它由应用服务和/或管理服务的创建、部署、激活、修改和任何相关操作组成。各APS支持如下功能：

1）联邦。依据共性高级目标，支持各域（如APS域或被引导的域）组合成一个较大的域（如被引导的域或两个组合的被引导域），同时维持局部自治。在APS联邦中，每个APS负责其自己的真实资源和虚拟资源以及由其监管的服务。在引

导联邦中，两个被引导的域联邦形成一个较大的被引导域。在这样的情形中，联邦应该考虑被引导域的目标，评估联邦的可行性。

2）协商。在有或无人类介入的情况下，协商可发生在自治实体之间。各 APS 和 PA 是可参与到协商中的主要实体。每个 PA 通告它提供的一个能力集（即服务和/或资源），用于 PP 的其他组件。各 APS 协商支持 SLA，SLA 是由被管引导域的运营商定义的。

3）分布。各 APS 提供通信和控制服务，使任务被分割成可同时运行在 PP 内多个 PA 上的其他任务。

4）监管。每个 APS 可以一种独立的、分布式的或协作式模式（即以联邦方式）操作运行。为确定物理资源或虚拟资源和服务是否需要重新配置，APS 收集合适的监测数据。高级目标、服务需求、语境、能力和约束应该被看作决策制定过程的组成部分。

5）系统视图。各 APS 负责管理系统视图，该视图使用 KP 进行存储和传播。各 APS 从各 PA 获取其操作运行所需的信息以及通过接口获取服务和资源。

6.1.3 相关工作

D. Clark [CLA 03] 提出构造能够“自管理”的新一代网络，给定高级目标，而不需要任何人类介入。Clark 的 KP 建议可被看作 Horizon 项目的管理、引导和各 KP 的合并。其他自治架构，如由摩托罗拉提出的 Focale [STR 06]，通过引入高级目标，扩展 KP 概念。在欧洲 FP7 项目中提出了其他架构：

1）自治网络架构（ANA）项目，在遗留互联网技术之外，探索了组织和使用网络的新颖方式 [ANA 11]。焦点主要在网络协议方面，而不在网络的管理和引导方面。

2）HAGGLE 项目处理自治机会性通信的一种创新范型 [HAG 11]。通过支持机会性联网范型，它开发了利用间断性连接的一种跨层网络架构。在这个项目中，在设备层实现引导，而在 Horizon 项目中，它引导整个网络。

3）BIONETS 讨论泛在计算的挑战 [BIO 11]，方法是通过一个自治的对等通信范型，适配社会的动态性，来处理异构性并取得扩展性。

4）CASCADAS 项目的目标是为自治状态感知的通信和动态适配服务开发组件型软件 [CAS 11]。其意图是针对自治和位置感知的通信，提出基于自组织分布式组件的一种创新架构。

5）Ambient（周边环境）网络（AN）[NIE 05] 是一个 FP6 项目，意图是为将运行在所有当前网络物理基础设施之上的无线和移动网络，开发一个软件驱动的网络空间基础设施。目标是使各设备能够相互连接，并相互通过连接到外部世界，同时提供无缝服务和漫游。

6）4D 是互联网的一种新架构模型，其中各任务被分成 4 个平面：决策、传

播、发现和数据［GRE 05］。在4D中，数据平面是一个简单平面，基于由决策平面接收到的配置执行动作。基于从发现平面检索到的信息，做出决策，其中发现平面构造一个物理资源视图。之后使用传播平面，将决策发送到数据平面。为说明该架构的优势，文章既没有给出仿真也没有给出实现；但是，作者论证，该架构的主要优势是将决策集中到单个平面，这消除了多个层处理类似争议的问题。在两个主要问题上，4D不同于Horizon。第一，4D将管理、控制和各PP融合到单个平面。但是，为使自治网络的设计成为一项可实现的任务，它没有描述一个框架或设计模式。第二，4D没有处理这样的事实，即不能依赖于单个管理实体，原因是每个域都是由一个不同组织机构运营的。但是PP考虑到这个事实，支持不同管理域的协商和联邦。

6.1.4 引导、管理和虚拟化平面的相互作用

Horizon自治管理架构模型由许多分布式管理系统组成，涉及4个平面：虚拟化平面（VP）、MP、KP和PP。这些分布式系统一起形成一个软件驱动的网络控制基础设施，该设施将运行在所有当前VN和物理基础设施之上，为各设备相互连接和相互通过连到外部世界提供一种方式，同时提供无缝服务。PP将通过行为与MP交互，各行为在6.3.2节定义。每个APS将控制一个或多个PA，且每个APS将处理与各PA（由该PA看管）互操作有关的引导问题。为支持这些任务，各APS将要求来自VP的信息，使用一些接口获取所要求的信息。进而各APS将支持服务部署，启动或关闭网络和用户服务。APS定义新服务部署上的约束，如将在其上安装服务的虚拟路由器或网络的集合，以及其执行参数中的一些参数。

6.1.5 在Horizon架构中引导平面的职责

PP的角色是监管，作为对变化的引导感知预警的响应和符合可应用的高级目标与策略，动态地调整和优化自治控制环。它监管和集成所有其他平面、行为，确保管理和控制操作的完整性。除了调整各PA的配置外，当需要时，PP也可启动和关闭各PA。PP可被看作一个控制框架，任意数量的组件可插入其中，取得所需的功能。对一个PP的需求，来自部署具有不同管理员或管理目标的几个自治控制环，在没有转换、协商、联邦和部署功能集的条件下，它们将不能互操作。因此，引导（平面）处理各APS的元管理，即自治管理控制环的部署和重新配置，以便支持它们的互操作。基于为每个管理网络域（形成被引导的网络）定义的高级目标集，做到这一点。PP确保管理系统的互操作，即使那些系统使用不同的高级目标和管理标准集也是如此。基于通用信息模型的不同数据模型之间的本体转换和映射技术，可被用来创建参与实体可协商、联邦等的通用语言。通过新SLA和策略的协商、冲突管理系统的去活，后跟其他管理系统的激活或在被引导网络内这种系统的迁移，可做到管理系统互操作的这个过程。整个引导过程为引导策略所监管，

这确定折中解决方法是每个被管理域将乐意为互操作做出让步。

6.2 引导平面功能和需求

PP与所有其他平面协同工作，因此它要求如下功能和需求：

1）考虑高级目标和顾客需要，排除低级技术和高级商务细节。

2）域内和域间引导所需知识，必须及时地由架构的PP和有关组件进行传播。

3）为取得所定义的SLA，各APS的响应时间必须被约束在某个边界内。

4）与各PA交互的接口，即处理环境变化和冲突。

5）支持各PA定义它们对其他组件和服务的依赖关系，以及它们的期望操作运行条件（如基于描述其所需服务和虚拟资源的策略）。

6）来自不同域的各APS的协作，要求使用开放协议和标准化的信息及数据模型。

7）使用VP和PP之间的特定接口和存储在KP中的信息，感知虚拟资源的状态。

8）来自不同PA中正交目标的冲突的解决方案。由此，PP必须能够达成妥协，使系统取得其目标。

9）所有PA的工作流控制，确保各PA的启动、初始化、语境化和关闭。为实施有用的功能，它应该控制一个PA触发其他PA的顺序和条件（即一个引导平面是各APS之间相互作用的模式）。

10）增强和演进。在不中断正常操作的条件下，为取得所需功能，PP应该支持相关数量的组件插入或退出。

6.3 初步引导平面设计

Horizon方法中的PPA概念由几个APS组成。每个APS将负责几个附属PA的交互。为确保端到端SLA和服务水平目标（SLO），在必要时，各APS将在其间交互。PP监管各APS的执行。它作为所有APS的一条控制工作流，确保各PA的启动、初始化、动态重新配置、适配、语境化、优化、组织和关闭。为实施一些有用的功能（即一个引导平面是各PA之间交互的模式），它也控制一个PA触发其他PA的顺序和条件。DP负责那些任务。最后，每个PA将与KP交互，获取有关被控行为的信息（见6.4节）。对每个PA，这个信息将是不同的，并将组成PA的共置视图。该共置视图定义一个自治组件操作所需的信息，且必须从这个组件收集信息。

PP组成一个或多个APS，每个域一个PP。每个受控PA将有其关联的行为。行为是各PA的一个封装器（wrapper），提供与PA交互所需的接口和功能。每个

被引导域有一个 APS，它与其他被引导域的 APS 通信达成协议，支持网络作为一个整体的操作运行。使用各 APS 之间的通信，PP 与其他 PP 的联邦是可能的。在这种情形中，来自两个或多个管理域的 APS 协商它们的联邦。将在 6.3.2 节详细讨论联邦、协商和监管的功能特征。

一个基于策略的管理系统从一个系统的功能抽象它的行为和动态决策。因此，一个系统的功能可以保持相同的，但其行为，特别就变化的语境条件而言，是可能变化的。在 PP 中使用的策略将其行为适配到变化的语境条件，如变化的商务目标、网络资源可用性或域成员关系。因为 PP 涉及分布式管理系统的引导，其测量直接与它们的需求（Horizon 项目中的各 PA）相关联。本质上来说，PP 各组件（即 PA）的行为，受到各分布式 PA 能力的支配。例如，单个 PA 可使用一个 APS 进行部署。这种部署的行为可能需要受到 PA“属主”（它将定义应该部署哪些 PA 的策略）和各 APS 的行为（是就与被部署 PA 的任何协作而言的）的支配。各 APS 应该实施 PA 为协商、联邦或分发而委派给它的策略。例如，一个 PA 可能将如下信息通知它相关联的各 APS，即与其资源有关的信息不应该输出给其他 PA，除非满足一些严格的授权要求。各 APS 也控制多个 PA，PA 直接作用于每个 VN 单元。这个组织结构的一个例子是切换（handover）过程。各 PA 可控制一个完整的决策，而切换决策各参数（即要被每个 VN 接收的最大客户端数和采用的认证过程）值的定义可由 APS 支配的高级目标所确定。

图 6.3 形象地给出了刚描述过的组织结构。高级 APS 控制低级 PA，并为那些组件设置策略和 SLA。各 PA 使用接口作用于虚拟资源，部署服务并执行虚拟资源和节点的 FCAPS 功能。最后，低级 PA 控制各控制协议操作运行所需的实时决策。如前所定义的，Horizon 架构有由引导策略系统持续地加以映射的几个层次的策略。PP 使用引导策略，这些策略是控制自治控制环的部署和联邦的高级策略。在一定意义上而言，引导高级策略是依据自治管理域的域间特征发挥作用的策略，控制两个 APS 何时和如何进行联邦。它们也支配过程如何解决冲突将由各 APS 实施，以及各 PA 在网络中的部署和分发。同时，APS 高级策略将焦点放在单个域的管理上。各 APS 从其相应的管理方接收引导的高级策略和 APS 高级策略。之后处理那些策略，且来自这项处理的有关信息被存储在系统视图中，并可在接到请求时由所需组件进行访问。例如，两个 APS 的联邦所要求的策略可由一个联邦行为加以请求，而与安全有关的策略可由自安全 APS 行为加以请求。在图 6.4 中给出 APS 设计的概略图。它支持由 PP 管理的系统中所有组件具有即插即用行为。各 APS 由三种类型的功能单元组成：

1）动态规划器：作为在一个 APS 中行为执行的一个工作流引擎。确定必须做到哪些行为，是必要的。

2）行为：实施特定的/个体的引导动作，这是一个 APS 实施所要求的。它们也代表网络中的特定管理任务。主行为是分发、联邦和协商。在下面各节详细描述

各行为。各行为作为 PA 的桩。它们也使用各 APS 特定的内部功能（核心行为），如各行为间的协商。

3）共置视图：是各 APS 到 KP 内部的“局部窗口”。视图由系统内视图和系统间视图组成。系统内视图提供单个 APS 内各组件看到的系统的一个整体视图。系统间视图提供各 APS 作为一个整体的总体的、复合视图，即它提供整体 PP 的一个视图。在本章中将进一步比较详细地概述这些组件的相应角色。

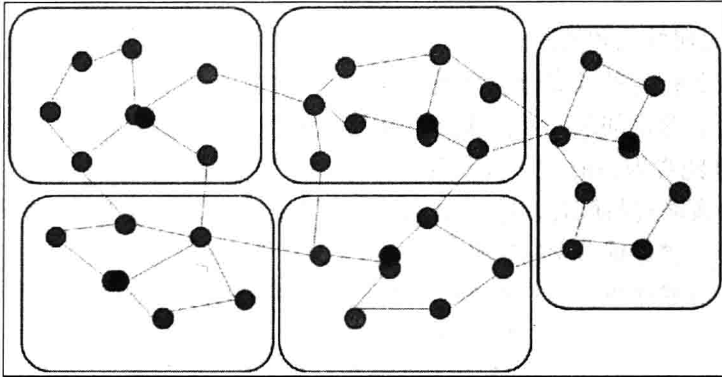


图 6.3 在 Horizon 架构中各 APS 的部署

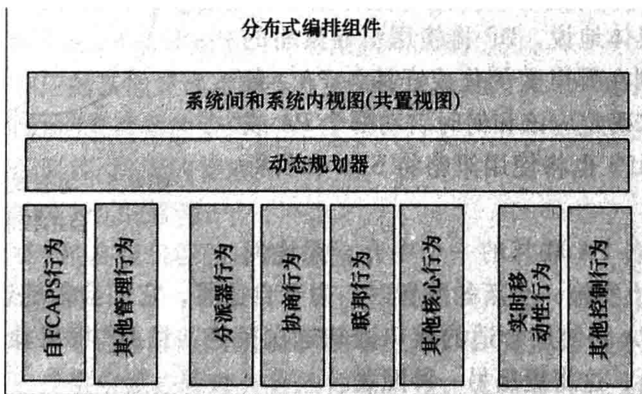


图 6.4 自治引导系统的设计

6.3.1 动态规划器

DP 作为各行为执行的一个工作流引擎。这样的控制包括如下任务：

1）为激活一个 PA，必须启动的各行为的顺序和条件定义（PA 组件的部署，定义它们对其他行为的依赖关系；如何和在哪里启动它们）。它作为所有 PA 的一个控制工作流，确保各 PA 的启动、初始化、语境化和关闭。为实施一些有用的功能，它也控制一个 PA 触发其他 PA 的顺序和条件（即一个引导是 PA 之间交互的

模式）。

2）监测行为，触发检查一个个体行为的操作没有与其他的操作冲突。

3）为每个被引导的 PA 和核心行为，转发所通告的高级策略，是由网络的运营商或属主提供的。

4）在行为的初始化和重新配置中冲突的识别。为重新配置部署规划，DP 启动协商行为。

5）采用一种即插即用（plug-and-play）、即拔即用（unplug-and-play）方法，触发各行为的初始化和关闭。

6）触发各行为的动态重新配置。

7）方便行为之间的交互。作为交互的一个例子，为解决配置冲突，协商行为使用 DP 与其他行为通信。那么，由 DP 触发一个重新配置行为。

DP 将由策略进行配置，这帮助 DP 选择必须被启动的行为（且由此涉及的服务和/或 PA）、服务或各 PA 参数与 SLA。

DP 的自治控制环（见图 6.5）是通过使用策略实现的，基于必须由虚拟资源满足的事件和条件，定义这些策略。无论何时满足这些条件，DP 都执行策略定义的动作，策略由一个或多个行为的启动组成。那些策略可在线改变，这支持 DP 重新配置自己和适应 SLA 中的变化。更具体地说，DP 将使用引导策略的分发策略来识别在哪里实例化或迁移各 PA。各策略也将被用来确定应该和何时启动哪个 PA 或核心行为。各 APS 也将使用策略将 SLA 和目标实施到各 PA。

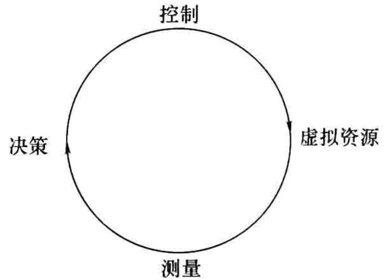


图 6.5　动态规划器的通用自治控制环

DP 也作为各 PA 联邦的一个中介。无论何时 DP 在各 PA 上实施一个新的配置时，由于自监管，它们会拒绝这个配置。在这种情况中，各 PA 将使用合适的接口通知这次拒绝，且 DP 将启动一个核心行为，它负责冲突解析，这将提出另一种配置。

6.3.2　行为

各行为是 APS 的子组件，它们实现某个引导任务。作为一个例子，APS 为高级策略的协商、任务的分发、服务和虚拟路由器的创建与销毁等实现各种行为。当必要时，各行为相互交互，即如果两个网络合并时，如果所需 QoS 不能得到满足，则联邦行为可与服务质量（QoS）行为交互。一个行为的生命周期受到 DP 的控制，DP 识别、启动和停止完成某项引导任务所必要的行为。尝试性地区分两种类型的行为功能单元（FE）：核心行为 FE（PP 的正确操作所需要）和 PA 管理行为 FE（与数据平面的控制和管理有关）。核心行为支持 DP 的操作，实现各 PA 的操作和

核心行为所必要的任务。核心行为 FE 的一些例子如下：

1）分发行为：将所需数据发送到不同的自管理引导行为 FE，并从之接收数据。

2）协商行为：实现在一个 APS 的各 PA 之间中介的算法，从而使各 PA 就共同目标达成一致。在协商完成之后，各 PA 在其相应管理域下应该采取自治和局部动作，以便集中实现这些协商的目标。也可在不同 PA 之间发生协商。

3）引导知识更新行为：管理与 PP 有关知识的传播。它控制与 PP 操作所需核心行为有关的信息和 DP 所需信息。

4）网络联邦行为：控制由不同 PA 控制的各 VN 的合并和隔离。这个行为识别组合/分解不同被联邦域的必要步骤，向 DP 建议（要实施）动作。

5）知识更新行为：管理 KP 的操作。它们定义“什么、何时和哪里”。要收集什么信息，何时收集和从谁（哪里）收集。这些行为是特定于服务的；但是，整个行为集合管理 KP 中信息的存储。每个 PA 要求预定义知识和运行时数据。映射逻辑支持数据（表示为一个标准化的信息模型）被转换为知识，并与由本体表示的知识组合在一起。

6）启动和初始化行为：在 DP 的管理下，启动和初始化其他行为。

7）重新配置行为：在 DP 的管理下，动态地重新配置和适配其他行为。

8）优化器行为：在 DP 的管理下，动态地优化和组织其他行为。

9）关闭行为：在 DP 的管理下，动态地关闭其他行为。

与提供实时响应的其他管理功能有直接互联关系的行为 FE 例子如下：

1）服务生命周期管理器的管理；

2）APS 分发/联邦/协商的管理；

3）各 APS 之间交互的管理；

4）与不同 APS 的高级目标的协商/分发；

5）各 APS 的监测；

6）由独立 APS 对网络所做改变顺序的网络一致性/完整性检查的管理；

在下面各节，提供主核心管理行为的详细描述，支持管理任务的联邦、协商、监管和分发。接下来比较详细地描述 PA 的行为。

1. 联邦核心行为

每个 APS 负责作为一个域进行监管的其虚拟资源和服务集合。联邦支持一个域集合被组合成一个较大的域，并将一个域分解为较小的域。当在服务之间监测到冲突且不能达成一个协议以及服务自己不能找到一个解析时，在寻找一个解决方案方面，联邦行为是重要的。建立联邦为三个不同属性。第一种情形是当 APS 可联邦网络时，方法是通过合并有冲突的两个网络而创建一个域。得到的域继承其组件的特点。在第二种情形中，APS 形成联邦的方法是，创建一个域，该域是被联邦网络的交集。新的网络需求是对这两个网络共同的那些需求。在第三种情形中，采用

这两个准则，各 APS 不能形成联邦。它们构造一个桥，支持这两个网络之间的连接。

在 Horizon 项目中联邦有两个功能特征。第一个处理所有参与联邦间的协议，第二个方便联邦间服务的提供。这两项功能特征是独立地处理的，但是相互依赖的，原因是在协议不能在联邦的参与成员之间达成的情况下，服务提供就不可能达成一致。主要挑战之一是，在互联网中，当商务和技术考虑需要以个体方式处理时，独立的管理域之间动态协议适配是困难的。在这种情形中，各 APS 应该开始一个特殊的协商行为，帮助新协议的协商，这将由各 APS 的决策所驱动。在 Horizon项目中，依据商务或技术考虑，由各 APS 组成的一个联邦是独立的。在下面各节概述商务或技术考虑。

（1）高级目标的联邦

当两个或多个 APS 需要有一个共同目标时，在 Horizon 中创建一个联邦。典型情况下，这个目标是在各 APS 的边界间提供保障可靠性的一个共性服务集合。每个 APS 将包含由策略连续体定义的其自己的商务目标。各 APS 需要确定可在联邦间维护的一个共性商务目标。一旦建立这个集合，就解决了联邦的商务方面。此外，任何成员可提出商务目标的修改。任何成员可选择离开联邦，离开一个联邦是各 PA 自监管功能的组成部分，如果它破坏了联邦的条款，则可能诱发惩罚。各 APS 可确定是否参与一个联邦，由此简单地建立有关联邦成员应该相互交互的某种共性理解。实际上，在一个联邦中提供和消费服务是一个技术问题，并必须独立地加以处理，以便使新服务能够被引入到一个联邦中。

（2）一个联邦的技术考虑

一旦一个 APS 参与到一个联邦，在联邦中它可消费并提供资源与服务。Horizon 项目中的各 APS 是面向如下目标的，即这些服务和资源使用的实际引导，以及加入和离开一个联邦的操作。各 APS 必须确保所请求的服务以这样一种方式变得可用，即遵守联邦所规定的各 SLA 和策略的条款。对于这一点，各 APS 必须能够评估各服务的配置，是否足以确保达成一致的联邦条款是否成立。APS 也必须遵守其自己的商务目标，并且如果这些目标不再被满足时，它可确定离开联邦。在离开一个联邦时，关联的 APS 必须将这个意图告知联邦成员。

服务和资源的不同类型要求不同类型的解决方案，以确保它们在一个联邦中被有效地使用。服务创建者的职责是确保新服务要在一个联邦内使用，它这样做技术上应该是可行的。

2. 分发核心行为

各 APS 提供通信和控制服务，支持各任务被分割成可并发地在一个 PP 内的多个 APS 上运行，甚至在多个 PP 间运行的部分。

（1）分发的商务考虑

商务考虑与每个 APS 是否要求信息、任务和代码的分发。信息分发是重要的，

因为一些 APS 可能产生对它们是独特的信息，并将需要进行显性控制。访问控制策略可有助于做到这一点。任务分发可被看作工作的分发，涉及在许多处理单元间的分布式处理。例如，一个特定视频的有效带宽的计算，可被分布在许多机器间，原因是这可能是一项耗时的过程。在这种情形中，商务考虑指这样的事实，即一个特定 APS 可请求其关联的 PA 分发一项处理任务。代码的分发稍稍不同于信息的分发，因为任务典型地并不预期从分发 PA 得到返回（结果）。代码的分发可以是一个 APS 升级所提供的一项特定服务的使能器，该服务提供依赖于许多 PA 的联邦。

（2）分发的技术考虑

可使用 KP 中与语境信息服务有关的概念，实施信息的分发。但是，需要牵涉各 APS，原因是它们可实施对信息分发的严格访问控制，从而遵守 APS 的高级策略。高级策略可指明信息是离开还是进入系统。从技术角度而言，任务的分发可由正在用来分发信息的平台来执行。但是，任务信息可以是一种特定格式的，其中可能需要严格的指令存在，指令该任务需求的各目标 PA（执行任务）。确定是接受还是拒绝正被分发的任务，是每个 PA 的事情。PA 需要将这个意图通知各 APS。之后，APS 实施分发，并通知 PA 这些意图是否得到满足。

代码的分发也可由用来分发信息的平台加以实施。在这种情形中，可执行代码是可被分发的一种特殊形式的信息。代码可被用来升级或部署一项新服务到其他 PA。因此，该 PA 指令各 APS，代码需要分发，且处理代码的分发是各 APS 要处理的事情。可使用服务使能器平面（SEP）来实施分发。

3. 协商核心行为

在针对未来互联网的一些方法中，必需的是，网络和服务提供商提供和发布他们的服务，从而之后可提供更复杂的服务。在 Horizon 项目中使这项需要得到满足的一个重要组件，是 PP 的协商组件。这个组件作为不同 PA 之间中介的一个虚拟服务代理，处理服务请求并提供支持，从而底层服务提供商可协商职责、任务和高级目标。考虑到底层服务提供商的特性、他们提供的服务、他们的关注点、他们的服务质量以及其他关键方面，应该实施这项支持。总之，应该提供这项功能，利用来自复杂的决策判定过程的服务请求者实体。

在 Horizon 项目中，每个 APS 通告它提供在 PP 中使用的一个能力集（即服务和/或资源）。协商组件使被选中能力的一个特定功能在各 APS 间达成一致。例子包括使用来自一个能力范围的一项特定能力（如当提供多项功能时的一项特定加密功能），当多个 APS 正竞争那项服务的独享使用时一项特定服务被授权专有使用，以及达成对要用的一个特定协议、资源和/或服务的一致意见。在 APS 和 PA 之间被引导的协商功能具有固有的商务和技术考虑。下面详细讨论这两个至关重要的方面——协商的商务和技术考虑。

当各 PA 在一个 APS 下协商高级目标时，或当不同 PA 与其他 PA 协商高级目标时，如果参与者将其内部商务目标与一个共性目标达成一致，则协商完成。那就

是说，当各参与者收敛到虚拟和物理资源的一个经协商的高级目标时，这些资源必须在它们相应的域（PA 和/或 APS）中被自治地分配、管理和控制。在商务目标的协商中，在协商过程中也要考虑职责、益处和惩罚。

值得指出的是，一个 APS 和/或一个 PA 可顺序地或并行地与几个 PA 和 APS 协商商务目标。接下来，商务目标的协商受到技术考虑的影响，是指各 APS 就资源方面达成妥协，来满足协商好的高级目标。一个 APS 的监管能力（和一个 PA 的监管能力）定义协商的原因和何时进行协商。协商能力确保各 APS 和 PA 总是与一个联邦中的其他实体协商。

各 PA 和 APS 可与几方具有活跃的经协商的协议。存在重新协商高级目标的一种潜在需要，这是由于承诺给这些目标的各资源中的统计变化（当目标不能得到满足时）或由于导致这种正确动作的一些内部决策。

当共性商务目标不能得到满足时，各 APS 也可触发重新协商。高级目标的重新协商是应该得到 PP 支持的一项功能。它是每个 PA 和/或各 APS 的监管能力的一部分，其中确定要重新协商其经协商的商务目标的哪个目标。重新协商可受到效用函数、成本/收益优化等的驱动。同样，在高级目标的重新协商过程中，也要考虑职责、益处和惩罚。

PP 为各 PA 和 APS 之间提供中介方式，从而使作为一个复杂服务请求的结果，它们可协商高级目标。因为各 PA 和 APS 可能属于不同管理域，所以它们可能讲不同语言，并可能以不同术语表示它们的高级目标。本体转换和映射技术可在参与实体上被用来创建通用语言。PP 必须提供要发生协商的机制，而不管这些技术特点中的任何一个是什么。

4. 引导核心行为

一个 APS 的监管功能处理自己关注的动作，它用之与其他各方（重新）协商高级目标，并采取可能涉及虚拟资源和物理资源承诺的动作，这可有助于在联邦间提供服务；每个 PA 可操作运行在一个个体的、分布式的或协作的模式。在每种情形中，为了确定它所监管的虚拟和非虚拟资源以及服务是否需要（重新）配置，PA 收集合适的监测数据。商务目标、服务需求、语境、能力和约束都被看作自己关注的决策判定过程的组成部分。APS 也可与其他 APS（系统间共置视图）联邦，因为它们是以自己关注的方式实施动作的，即它们可具有自我监管的性质。

下面介绍监管的商务和技术考虑。

在一个联邦环境中，各 APS 以如下一种方式提供支持，即它们的底层 PA 知道一个联邦内其他 PA 的需要。APS 可确定它将与哪些 PA 协作。如前面介绍的，每个 APS 可以一种个体的、分布式的或协作的模式操作：

1）各 APS 作为个体实体，当它们不是任何联邦的组成部分时，独立地和自治地工作，并监管其自己的虚拟资源和物理资源以及服务，这些是由其自己的商务目标驱动的。它既不与其他 APS 共享共同目标也不共享共同职责。各 APS 应该处理

和传输由相应 PA 发送到它们所处联邦中其他 PA 和/或 APS 的消息。

2）作为 PP 分发功能的一个结果，各 APS 可与一个联邦中的其他 APS 一起工作，其中复杂服务是在一个协商阶段之后，协作每个分布式 APS 的活动、资源和服务进行提供的。每个 APS 应该向各 PA 提供许多契约接口（contract interface），PA 使用它们促进和中介一项复杂服务的协商、它的激活和维护。这些接口可被解释为契约接口，接口隔离各 PA 的内部结构和能力。

3）当一个 APS 作为一个局部或一个全局协调器时，它以一种协作模式工作。当各 APS 将其控制的一部分委派给一个 APS 时，一个协调器负责在一个给定 PP 中协调其他 APS 的功能。当一个 APS 在不同 PP 间协调 APS（即 APS 间）的功能时，它作为一个全局协调器。这就使模拟客户端—服务器、n 层、集群和对等架构成为可能。

不管操作模式为何，PP 都应该为一个 APS 通过一个良好定义的接口集合，提供监管其虚拟资源和非虚拟资源的方式。各 APS 的目标是使各 PA 总是能够依据自己关注的策略，决定要采取的动作，策略受到商务目标和能力的驱动。其决策也应该考虑参与到联邦中其他 APS 的策略。但是，APS 可拒绝遵守联邦的某些策略，此时那些策略与其自己的商务目标是不一致的。在这种情形中，重新协商或改变联邦的策略是 APS 的职责。

5. APS 行为

各 APS 使用 APS 行为作为与各 PA 通信的一个接口。这样的封装使各 APS 能够以一种统一的方式看到各 PA，具有与核心行为相同的接口。各 PA 和 APS 之间的通信遵循一个确定的信息模型。进而，一旦 APS 行为提供从一个 APS 的特定设计问题到各 APS 的操作运行哲学，则它们是 CPA 的包装器，这类似于远程过程调用（RPC）或 CORBA 中的桩和骨架。这样一个包装器的用途之一，是将 APS 的不同实现对各 APS 是隐藏的，如确保单个通信点，即使 APS 可能是分散于几个虚拟节点的一个分布式组件时也是如此。

包装器也定义各 APS 必须支持的一个命令集，这使各 APS 能够编排它们的联邦和分发（distribution）。接口为各 PA 传播和重新协商与各 APS 的策略提供机制，这使 PA 为自我监管的。各 PA 也支持虚拟资源和协议的准实时控制。高级自我 FCAPS APS 主要关注于资源的长期管理，而部署各低级 PA 是为了尽可能快地对 PA 感知语境中的变化做出反应。由此，各低级 PA 使用简单算法，依据一个预确定的总体目标执行动作。那些目标受到 APS 所定义各策略的控制支配。由于对其反应的时间约束，那些 PA 将在单个虚拟资源或一个小型虚拟资源集合上执行动作。在某个意义上，各低级 PA 可被看作一个自治系统的第一个控制环，依据预确定的目标和没有任何内嵌学习能力的基础上执行动作。可使用的可能技术例子有状态机、比例—积分—微分（PID）控制器、模糊决策等。各低级 PA 与网络 SLA 定义的一个 QoS 的维护关联，控制虚拟节点的参数及其运行算法，如移动性管理算法。

6.3.3 系统内和系统间视图

各 APS 使用 KP 存储和分发其操作运行所需的信息。依据信息与一个给定 APS 的相关性，信息可被分成两部分或视图。

系统内视图与编排引导域内各服务所需信息有关，而系统间视图处理几个引导域的引导。系统内视图包含使各 APS 能够了解特定位置的信息。系统间视图为协作各 APS 提供类似信息。

由此，系统内视图处理运行在一个 APS 边界内的各项服务，拥有与 APS 操作运行相关的信息。这个视图也被称作一个共置视图。实际上，取决于馈入的算法，共置视图一定是不同的。例如，不仅可定义一跳或两跳或一个完全的共置视图，而且可定义更复杂的共置视图，如图 6.6 所示。

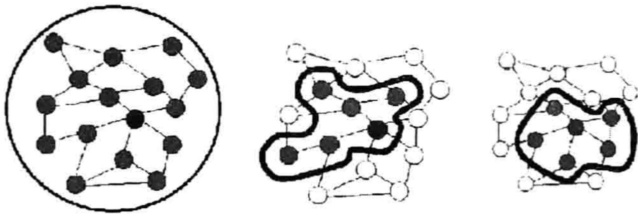


图 6.6 共置视图的不同解决方案（整体的、中间的和一跳的）

在共置视图的定义上存在一个重要的折中：大型共置视图可使用系统的整体状态的描述进行决策。但是，传播和处理成本，如所有相关节点的知识，是较高的。由此，一个最优的共置视图应该基于要求解的问题和清晰的性能度量元上进行定义。

各 APS 维护共置视图，定义哪些信息必须被存储在这些视图中，针对哪些虚拟节点或资源，以及必须以多大频率进行更新。在下一节描述 PP 与 KP 的接口。

6.3.4 APS 的接口

本节描述各 APS 与 Horizon 架构其他单元的交互，如图 6.7 所示。首先描述各 APS 与 KP 的接口。下面介绍与虚拟资源、各 APS 和服务使能器平面的接口。

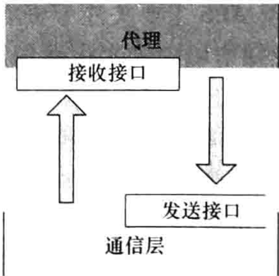


图 6.7 APS 的接口

KB 是一个存储库，其中管理和控制任务所需的所有信息和知识都保存在其中。因为每个单元应该能够自治地工作，所以 PP 应该能够自治地提供所需的信息。PP 向各 APS 提供用于监测的所需参数值，以及应该从网络中的哪里采集信息。这些参数值可由动态规划器确定，依据 PP 的功能和网络的状态，且这些值提出专用于这项任务的一项特定

行为。应该考虑 KB 中的两种类型知识：与管理任务有关的那些知识和与引导任务有关的那些知识。

PP 的目标之一是识别自治决策所需的信息，接下来是激活所需的 KP 功能，这将确保这个信息的及时收集和交付到各 APS 或一个特定的行为。KP 和 OP 之间的交互要求如下功能：

1）Lookup（Info）：从 KP 请求某片信息（或知识）的查找。这用于获取被引导 APS 的策略、SLA 以及有关的配置参数。

2）Store（Info）：在 KP 上存储信息。这个功能用于 OP 中产生的知识，之后存储在 KP 上。

3）Subscribe（Event，Component）：由 KP 的被存储信息上的一个条件定义，这个函数支持将网络中发生的有关事件通知给各 APS。事件可定义条件和可在其上发生这些事件的节点或 APS 集合。例如，各 APS 可关注于监视某些故障的发生，触发负责故障处理的一个 APS 的重新配置。该订阅也识别哪个 APS 和它的哪个组件将处理该信息。

4）Watch（Info）和 unWatch（Info）：由各 APS 使用，用来定义哪些信息必须或不必周期性地由 KP 分别进行收集和传播。例如，可监测与某个 PA 相关联的一个网络接口的平均带宽利用率，以便验证这个组件是否应该被迁移到网络中的另一个节点。获取的信息由信息类型、其共置视图（必须从哪些虚拟节点收集）、这种更新的周期性以及要求这个信息的组件等定义。一旦各 APS 发出一条 Watch 请求，KP 就负责该信息的维护，将之直接交付到涉及的组件。Watch 使用的一个例子是当一个 APS 定义活跃流的数量时的情况，它应该与自治性能优化 PA 通信（传递给 PA）。当不再需要这个信息时，各 APS 使用 unWatch 功能释放 KP 资源。

5）Push（Information，PAs）：用于在各 PA 间的同步消息传递。PA 使用各 APS 的 Subscribe、Watch 和 Push 接口，创建它们的系统间视图，用来馈入动态规划器和核心行为，其中带有其正确的参数值。那些功能也被用来进行 APS 间通信，因为一个 APS 可使用相同的功能监视其他 APS 的状态。访问来自其他 APS 的信息应该受到访问策略的控制，因为各 APS 可能受到不同组织机构的控制，由此各 APS 也许希望隐藏其网络管理的一些方面。

6）接口 SEP/APS、虚拟资源/APS 和 PA/APS：如前所述，PP 的最终任务是使用网络上的各 PA。各 PA 的部署应该在考虑服务的管理需求下完成。PP 使用 SEP 实施各 PA 的部署。但是，PP 使用 SEP 来使用各 PA，它不处理服务的部署。PP 需要与虚拟资源交互，为各 PA 行为的部署，将有关网络状态的物理管理和控制信息提供给 PP。各 APS 使用的最后一个接口是各 APS 与各 PA 的接口。这个接口实际上是由 PA 包装器行为实现的。

6.4 引导代理

本节描述分布式引导代理平面。这个平面由分布式智能代理组成，每个代理与一个 NE 或一个 NE 集合相关联，如图 6.8 所示。

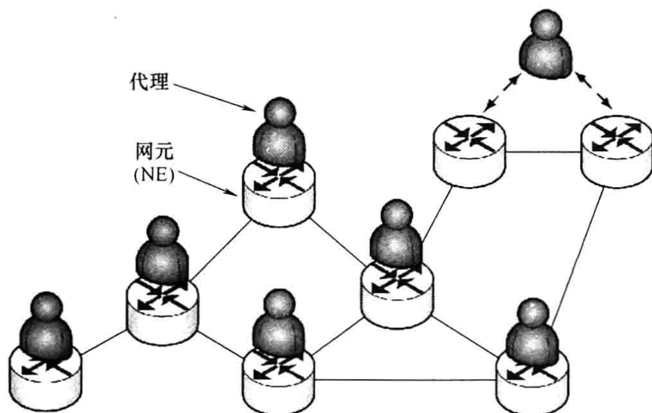


图 6.8　形成引导平面的智能代理

通过将代理分布在网络间，PP 使用户能够局部地处理问题。实际上，相比那些由远程争议导致的问题，局部问题处理起来经常是比较简单的和容易的。此外，在早期一个局部问题可采用局部方式解决，而不是以一种中心式方法解决。例如，一个代理可立刻改变其 NE 的配置，以对一个局部负载问题做出响应。除了纯局部问题外，各代理可协作处理这个邻居关系范围中的问题，例如一个连通性问题可能被几个代理检测到。之后各代理可协作更准确地刻画问题，并最终向较高层提供一个解决方案或报告。为实施任务，各代理可使用功能强大的分布式 AI 技术。作为 PP 组成部分的 KP，由各代理的共置视图组件组成。类似地，OP 由一个动态规划器和不同行为组成。引导系统的架构如图 6.9 所示。

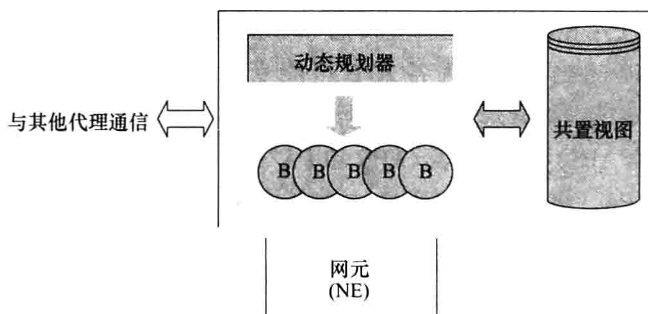


图 6.9　代理架构概图

PP的第一部分由一个KP组成，它是内嵌于各代理的一个分布式信息库。它包含各代理的局部信息和网络的全局信息。在直接从其各NE的局部观察和间接地从网络其他部分（通过与其邻居交换信息）得到的信息基础上，每个代理维护其自己的网络视图。网络的这个以代理为中心的视图，焦点是附近的（close）网络环境，称作共置视图，如图6.10所示。

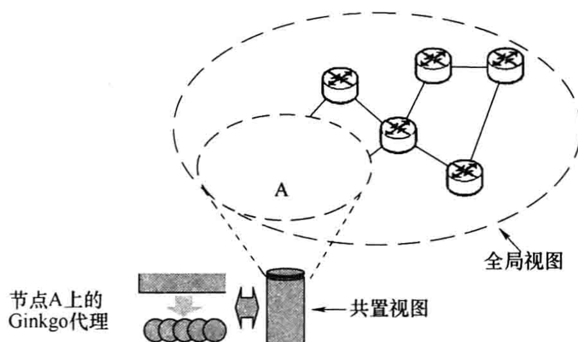


图6.10 每个代理有其自己的网络共置视图

共置视图的理论依据是，发生在一个代理邻居范围中的事件，一般来说，要比发生在网络的一个远端部分的事件，对代理有较大的重要性。较早地知道局部事件和更准确地记录在共置视图中的事实，使代理快速地和合适地做出响应是比较容易的。各代理定期地检查出现在其共置视图和网络环境中的重要变化。它可确定自动地调整其被管NE的某些参数或请邻接的代理调整它们相应的NE参数。共置视图的使用驱动代理之间的隐性协作，通过它们共享的知识而相互“影响”。隐性协作是PP中各代理间协作的主要模式。这种模式的协作是简单的，特别是鲁棒的和良好地适合于动态改变的环境，原因是它不要求建议代理之间的一个显式对话和严格同步。

一个自治系统由不同自治单元组成，它们协作取得自治系统的整体目标。因此，当到达一个阈值以实时地确定要做什么，要求一个OP。作为变化语境的响应，并符合适用的目标和策略［TSE 09］，OP负责系统行为的监管和集成过程。通过行为和动态规划器，自治架构实施这个过程。一个代理能够做的事情被定义为一个行为集合（图6.11中的“B”）。这些行为中的每个行为可被看作具有一些专家能力的一个专用功能，处理要由该代理实施工作的特定方面。

下面列出典型的行为种类：

1）与其他代理协作，更新共置视图。

2）个体地或集体地推理，评估状况，以确定应用合适的动作。例如，一个行为可简单地负责一个NE上的计算带宽可用性，它也可定期地实施一个复杂的诊断场景，或它可专用于特定网络条件的自动识别。

3）作用于NE参数。例如，一个行为可调整一个DiffServ语境中的QoS参数。

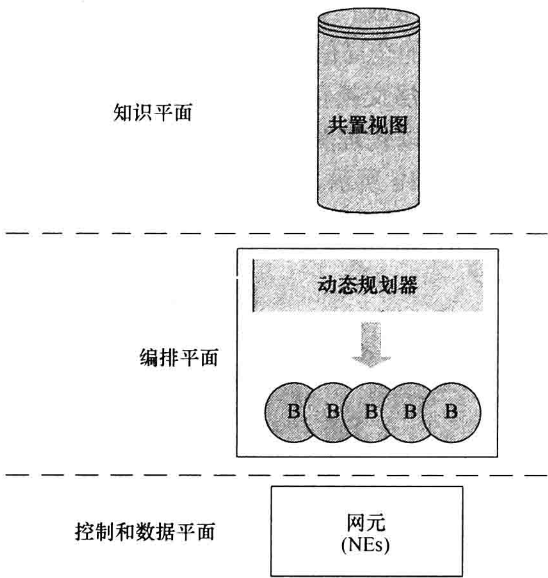


图 6.11　引导系统的概图

各行为可访问在每个代理内操作的共置视图，作为代理的行为间共享的一个白板。在一个代理内行为的激活、动态参数化和调度，由动态规划器实施。动态规划器确定哪些行为必须是活跃的、何时它们必须是活跃的以及采用哪些参数值。动态规划器检测共置视图中的变化和外部/内部事件的发生。从那里，它编排各代理对网络环境中变化的响应。

6.5　测试床

重要的是在一个真实的网络中部署系统之前，评估所提出的 APS。图 6.12 形象地给出采用这个目标构建的一个测试床［SEN 11］。

测试床包含两个 VN：VN A（虚拟的 A）和 VN B（虚拟的 B）。这两个 VN 每个都包含两个虚拟路由器。虚拟路由器位于真实的主机 zeus 和 dionisio 处。对于要被实例化的 VN A，需要在真实主机 zeus 上实例化虚拟路由器 horizonzeusA 和在真实主机 dinisio 上实例化虚拟路由器 horizondionisioA。对 VN B 需要类似的实例化。

创建 VN A，通过一条两跳虚拟路径，互联主机 artemis 和 apolo。类似地，创建 VN B，互联主机 nix 和 cronos。虚拟路由器之间的虚拟链路可被映射到物理路由器 zeus 和 dionisio 之间的一条 100Mbit/s 链路或一条 1Gbit/s 链路。最初，这两条虚拟路径共享 100Mbit/s 链路。

实施了初步的试验，确认在两个 VN 的操作运行期间改变虚拟路径映射的可能性。思路是过载 100Mbit/s 链路，之后，仅将一条虚拟路径迁移到 1Gbit/s 链路。

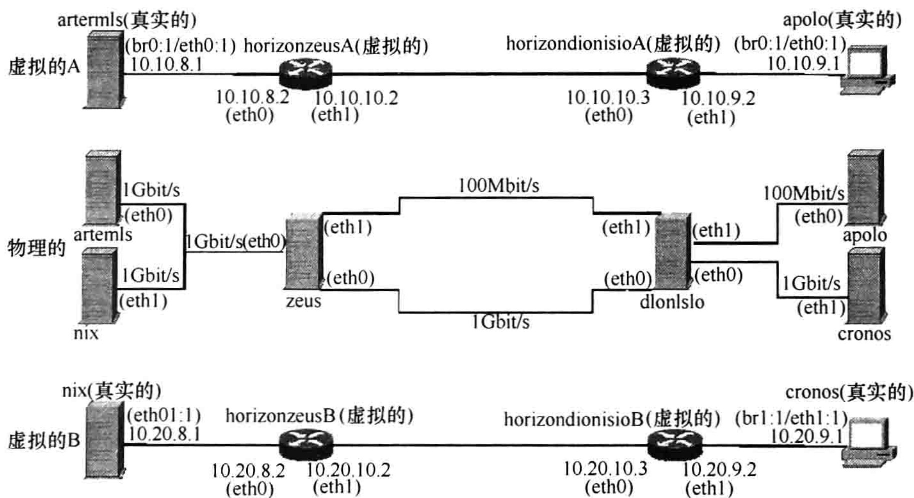


图 6.12　测试床

这个试验仿真这样一个场景，其中位于路由器处的各 PA 检测一条链路的高利用率，并做出迁移虚拟链路之一的决策。

下面各节给出有关测试床的更多细节。6.5.1 节概要描述用来构建测试床的一些工具，而 6.5.2 节给出初步试验的结果。重要的是指出，这些试验是手工实现的，没有来自所提供 APS 的各 PA 的中介在内。在 6.6 节评述多代理 APS 的实现，在 6.7 节给出采用各 PA 得到的结果。

6.5.1　工具

本节描述用来构建测试床的主要工具，在试验中要使用这些工具。使用这些工具进行各 VN 的部署和操作，且 Ginkgo 平台被用于多代理 APS 的开发。

1. qemu

qemu [BEL 05] 是一个处理器模拟器，也可用来创建虚拟机（VM）。相比于其他虚拟化平台，qemu 具有容易安装的优势，因为对于主机操作系统而言，它是像任何其他应用一样的一项应用。劣势是 qemu 容易受到主机操作系统相同进程调度和内存管理算法的影响。

qenu 是通用公开许可证（GPL）下免费软件的一部分且是开源的。存在一些额外的组件，可与 qemu 一起使用，来改进寄居操作系统的性能，如果真实处理器包含被模拟处理器的相同指令。思路是允许寄居操作系统直接访问真实处理器。

2. KVM

基于内核的虚拟机（KVM）[KIV 07] 是基于机器模拟器 qemu 的一个全虚拟化 hypervisor。它运行在 x86 架构上，采用像 Intel VT 和 AMD- V 等新的虚拟化技术。KVM 由 Linux 的一个内核模块组成，当载入时，在/dev/kvm 处提供到寄居 VM

的初始化和控制的一个接口。

KVM 是 GPL 下的一个免费软件且是开源的，这使得能够使用外部工具（像 Libvirt）来控制它。

对 Horizon 项目有益的一些 KVM 功能特征包括：

1）全虚拟化中的良好性能。

2）各 VM 的实况迁移。

3）支持 SMP 主机和寄居机。

4）内存气球法。

5）采用桥接、路由或专用网络的 VM 联网。

在性能方面，KVM 超越了 qemu，因为 qemu 的主要目标是一个处理器模拟器，而不是一个虚拟化平台。但是，qemu 具有运行在任意架构上的优势，且容易安装，因为不需要采用特定内核或模块来修改主机操作系统。

在本节给出的多代理 APS 的初步试验，是在使用 qemu VM 增强的一个测试床中实现的。在 6.7 节中给出的多代理 APS 的最终试验，是在采用 KVM VM 的一个当前测试床上实施的。

3. Libvirt

Libvirt [COU 10] 是访问 Linux 虚拟化能力的一个 API，支持各种 hypervisor，包括 qemu、KVM 和 Xen 以及来自其他操作系统的一些虚拟化产品。它支持各 VM 的局部和远程管理。采用 Libvirt，则独立于虚拟化平台，一个 PA 使用相同代码监测和控制虚拟资源是可能的。

Libvirt 是以 C（支持 C++）实现的，并包括对 Python 的直接支持。它也支持许多语言绑定，这些是针对 Ruby、Java、Perl 和 OCaml 实现的。在试验中，使用了 API 的 C 版本和 Java 绑定。

Libvirt 对这项工作的重要性是使虚拟化技术是独立的，这有利于未来的设计改变。采用 Libvirt，则在不修改各 PA 的条件下，将虚拟化平台从 qemu 改变为 KVM，将是可能的。

4. Ginkgo 分布式网络引导系统

Ginkgo 分布式网络引导系统 [GIN 08] 是基于自治网络的一个代理平台。它有针对计算机网络的一个引导系统的开发所用的构造块。该框架支持创建轻量的和可移植的代理，这在异构环境中有利于它的实现：路由器、交换机、主机、有线网络和无线网络。各代理扮演自治计算的自治管理器的角色。在分布式管理器靠近其被管单元的情况下，可以局部方式进行监测。

该平台支持代理集群的创建。邻接代理交换信息并得到网络的一个共置视图。这个信息被存储在 KB 中，该 KB 使用一个信息模型，这方便代理之间的通信。策略文件是另一个库，包含应用的规则。在拥有环境和应用的知识情况下，多代理系统可向网络提供一个自知识（self-knowledge）性质。

各行为实施代理的感知、认知和动作。它们向 KB 提供输入，感知和预测有威胁的事件，并在被管单元上实施改变。各代理也有一个动态规划器，它可改变行为的参数并控制代理的生命周期，依据的是 KB 中的信息和策略文件中的规则。这使自配置、自愈、自优化和自保护功能的开发成为可能，提升了自我管理能力。

Ginkgo 平台提供了实现各 PA 所必要的关键功能，这在 6.4 节中已给出。

6.5.2 测试床中的试验

实施了初步试验，评估在各 VN 操作运行过程中改变虚拟链路映射的可能性。本节描述在这些试验中得到的结果。

虚拟路由器是基于 qemu 版本 0.12.5 的 VM。为创建和操作虚拟路由器，使用了 Libvirt 版本 0.8.3。物理机器和 VM 都使用操作系统 Debian GNU/Linux，内核版本为 2.6.32。使用以太网协议（802.3）通过虚拟接口和网桥，在数据链路层上创建虚拟链路。各网桥受到 Bridge-util 软件包的 brctl 工具（版本 1.4-5）的控制。

初始情况下，各 VN 的虚拟链路共享基层网络的 100Mbit/s 链路。在两台 VN 中使用 iperf 产生用户数据报协议（UDP）流量，直到 100Mbit/s 链路饱和为止。此时，在 VN 的路由器上手工地执行脚本。

图 6.13 画出两台主机 cronos 和 dionisio 之间的网络时间（RTT），这两台机器使用饱和的 100Mbit/s 链路由 VN B（图 6.12 中的虚拟的 B）连接。在时刻 40s 处，执行将虚拟链路的映射从 100Mbit/s 改变为 1Gbit/s 的脚本。可观察到这次改变使这

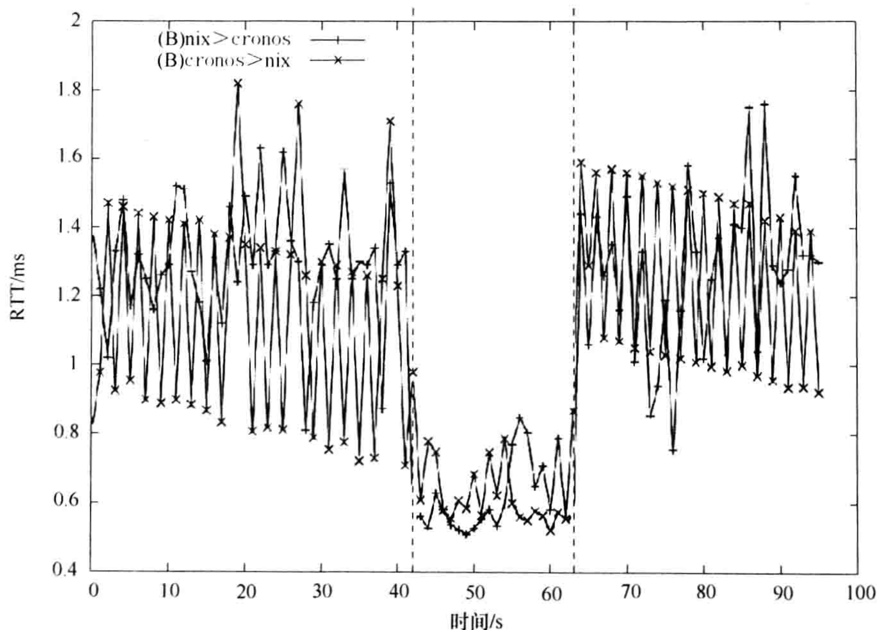


图 6.13 虚拟网络 B 的各主机之间的 RTT

两台主机之间的 RTT 值降低的结果。在时刻 60s 处，执行脚本，返回到 100Mbit/s 链路的虚拟链路映射。结果 RTT 增加到其初始范围值。

在 6.7 节将描述一个试验，它类似在本节中摘要描述的试验。区别在于前者采用各 PA 采取动作的情况下实施的，而不是手工执行脚本。

6.6 多代理 APS

本节在测试床内实现一个多代理 APS，实施 VN 的管理［SOA 12］。各 PA 运行在网络的各节点上，如图 6.14 所示。APS 中的各 PA 应该监测和控制物理资源和虚拟资源。采用来自 Ginkgo 平台版本 2.0.13 的支持，开发了 APS。采用 Java 版本 1.6.0-21 编译和执行各 PA。对于由各 PA 实施的 VN 监测，使用 libvirt-java 库版本 0.4.6。

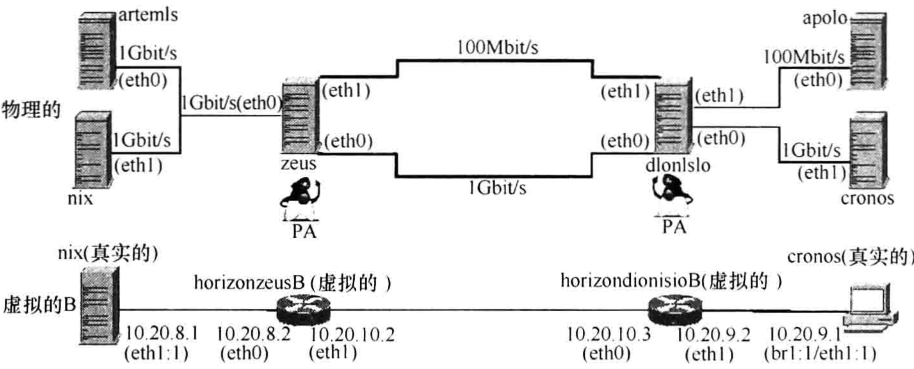


图 6.14 物理路由器中的各 PA，它们管理虚拟路由器

在 KB 中，每个网络资源遵循一个信息模型。每个资源受到单个 PA 的控制，因此，在 KB 中这个资源的每种表示都是唯一的。在信息模型中，链路是有向的，即每条线路（wire）都表示为相反方向的两条链路。各 PA 位于路由器中，且它们在一条直接链路上发送数据，监测和控制该链路。依据参考文献［FAJ 10］，图 6.15给出信息模型的类图。

邻居关系由域内各路由器的所有 PA 组成。Ginkgo 平台中的所有通信都基于 KB 中存储的信息的传播扩散。以一种同构的和有结构的方式，KB 存储由各 PA 使用的数据（节点的拓扑信息、状态等），并提供这个知识在各 PA 间的传播扩散机制。一个 PA 有其自己环境（共置视图概念）的视图可直接采用 KB 实现。在一个 PA 内，该 KB 可被看作一个共用的黑板，可由各行为访问。那些行为实现自治处理，并作为有机构成组件，永久地将它们自己适配于环境变化。这些组件的整体功能是由动态规划器编排的，后者遵循由应用程序员提供的一个策略。动态规划器操作存储于 KB 中的有结构的知识。

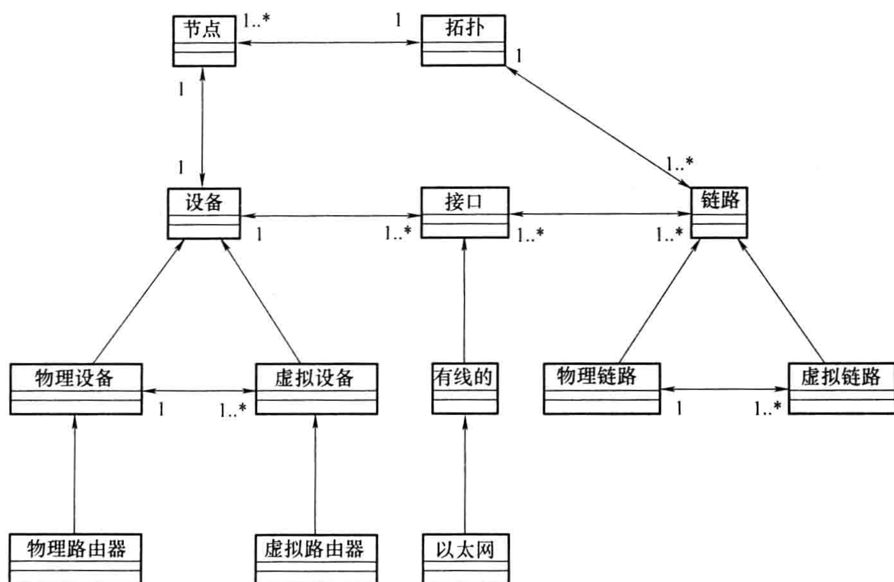


图6.15　各PA的信息模型

在这个实现中，各PA有四种行为：Monitor、Analyze、Plan和Execute，它们是以顺序方式周期地执行的，形成管理器的自治周期。下面列出它们的动作：

1）Monitor（监测）：通过libvirt收集来自网络接口的数据，并馈入KB。

2）Analyze（分析）：评估物理网络链路的使用情况。如果利用率超过策略中定义的阈值，则在KB中更新物理链路信息，通知其他行为。

3）Plan（规划）：当分析行为表明一次威胁，即它选择另一条物理链路（如果有的话）来接收一条虚拟链路（映射到过载的链路）的一条流，此时规划动作。

4）Execute（执行）：执行动作，通过KB，将变化通知其他行为和各PA。

策略文件有一个规则集，在每个PA周期都要对之进行评估（见图6.16）。各规则有一个名字和一个条件表达式。规则RINIT（行2）使用impulse init条件，一旦处于PA执行的开始，它就是真的。在这条规则中，采用start原语配置和启动各行为。遵循策略文件，在KB中设置和得到值，也是可能的。规则RINIT中的第一个动作是将control个体对象的speed属性值设置为1.0（行3）。这个值代表以秒为单位表示的各行为周期的当前时长。这个时长采用changerate原语定义。行为调用顺序可由其优先级控制，采用changeprio原语定义。采用setcontrol原语为这些行为配置参数。动态规划器应用策略并将threshold属性设置为0.5（行7）。这个值表示物理链路使用的一个百分比。当Analyze行为打开speedup标志和当前speed是1.0时，规则RFASTER（行17）将各行为周期的时长改变为0.2s。当Analyze行为关闭speedup标志且当前speed是0.2时，规则RNORMAL（行23）将这个周期时长设置回1.0s。

```
[frame=lines,fontsize=\small,numbers=left]
(policy (subgoal main (rules
  (rule RINIT if (impulse init) (
    (set control.speed 1.0)
    (changerate MonitorBehavior 1.0)
    (changeprio MonitorBehavior 3)
    (start MonitorBehavior)
    (setcontrol AnalyseBehavior.threshold 0.5)
    (changerate AnalyseBehavior 1.0)
    (changeprio AnalyseBehavior 2)
    (start AnalyseBehavior)
    (changerate PlanBehavior 1.0)
    (changeprio PlanBehavior 1)
    (start PlanBehavior)
    (changerate ExecuteBehavior 1.0)
    (changeprio ExecuteBehavior 4)
    (start ExecuteBehavior)))
  (rule RFASTER if (and control.speedup (= control.speed 1.0)) (
    (set control.speed 0.2)
    (changerate MonitorBehavior 0.2)
    (changerate AnalyseBehavior 0.2)
    (changerate PlanBehavior 0.2)
    (changerate ExecuteBehavior 0.2)))
  (rule RNORMAL if (and (not control.speedup) (= control.speed 0.2)) (
    (set control.speed 1.0)
    (changerate MonitorBehavior 1.0)
    (changerate AnalyseBehavior 1.0)
    (changerate PlanBehavior 1.0)

    (changerate ExecuteBehavior 1.0))))))
```

图 6.16 策略文件的规则

6.7 结果

评估了依据变化的环境，在各 VN 上自动适配（由多代理 APS 实施）的时间和报文丢失。这些动作类似于 6.5.2 节中给出的那些动作。但是，在这个试验中，虚拟路由器是基于 KVM 版本 0.12.5 的 VM。为创建和操作这些虚拟路由器，使用 Libvirt 版本 0.8.3。物理机器和各 VM 都包含操作系统 Debian GNU/Linux，内核版本为 2.6.32。通过物理节点内虚拟交换机中的虚拟接口和规则，在数据链路层创建虚拟链路。使用 Open vSwitch [PFA 09] 版本 1.3.0 作为虚拟交换机，并使用 ovs-ofctl 工具控制各条流。

在策略文件（见图 6.16）中给出各 PA 的配置参数。在正常状态中，自治周期的行为是以 1.0s 的间隔运行的，此时网络中所有物理链路的利用率都低于 Analyze 行为中设置的 threshold（配置为 50%）。当发生一次过载时，周期频率增加到每秒

运行5次的一个速率，即间隔为0.2s。

图6.17给出整个试验过程中物理链路的利用率。由iperf应用产生的一条UDP流，以恒定速率70Mbit/s传输通过VN A，在5.0s时开始传输。快速以太网物理链路达到70%的利用率，超过在Analyze行为上定义的threshold，则Plan行为开始实施纠正动作。PA用掉1.0s时间识别到这种状况，并对网络实施调整。

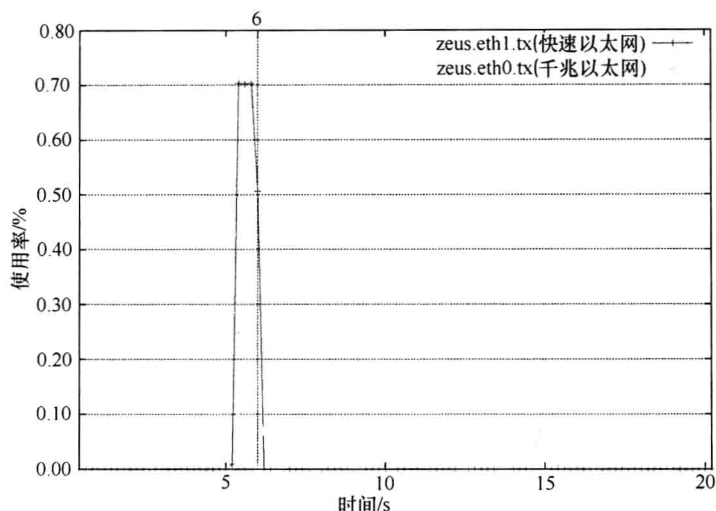


图6.17　物理链路的利用率

图6.18给出VN A上的流量。如图所示，视频没有影响VN吞吐量。这意味着，将虚拟链路迁移到一条新的路径不会诱发报文丢失。在这个场景中，虚拟节点

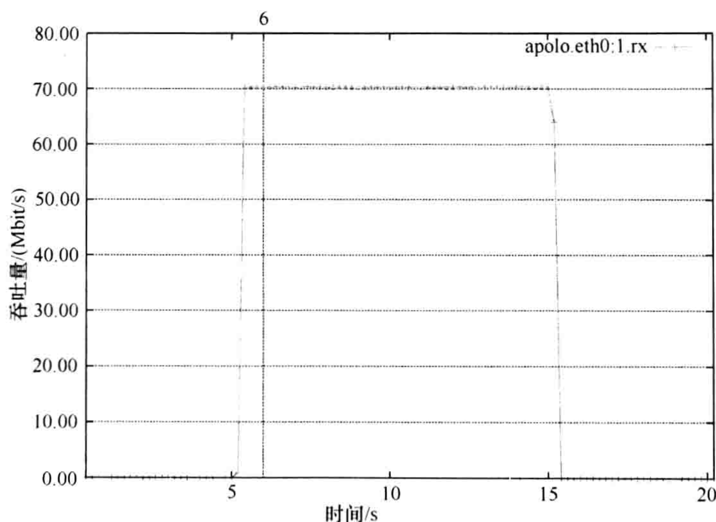


图6.18　虚拟网络A中的流量

horizonzeusA 和 horizondionisioA 之间的虚拟链路，被映射到物理节点 zeus 和 dionisio 之间的单条物理链路上（见图 6.14）。如果在路径上存在其他基层节点，则多代理 APS 将必须做出路径上的后向适配，以确保没有报文丢失。

6.8 用于虚拟网络自管理的多代理系统

本节介绍自治计算技术如何可被应用到 VN 的管理。它给出一种分布式架构，在一个基层网络（作为 PP 的组成部分）之上支持 VN 的自管理。NE 的自治管理器维护监测、分析、规划和执行的一个控制闭环，为下一次迭代向一个 KB 提供馈入。焦点是 VN 的自愈，但 VN 自管理的分布式架构足够通用，可由自治计算中的其他功能使用，如自配置、自优化和自保护。在下面也介绍了相关工作，给出这项工作主要贡献的语境。

在参考文献［HOU 10］中给出资源失效和严重性能降级情况时，维护 SLA 的一个多代理系统。基于物理节点的相似性，各代理形成组并管理之。也使用相似性功能，来选择在出现故障时恢复虚拟节点。架构也基于多代理系统，且通过从备份内存恢复虚拟路由器，实现 VN 的快速恢复机制，以降低路由协议的收敛时间。

在参考文献［MAR 10］中，给出一个分布式管理架构，自组织 VN 的目标是维护物理资源的有效利用率。用于自组织的算法基于自治控制环。它监测目标链路，并通过迁移虚拟节点，尝试最小化网络中的流量负载。给出类似架构，并实现一个原型，焦点放在 VN 的自愈上，以此评估所提出的建议。

6.8.1 原型实现

图 6.19 形象地给出建立的测试床，用来测试在一个受控环境中的系统。网络核心由 4 台机器组成：zeus、atlas、dionisio 和 cronos。机器 zeus 和 dionisio 也通过千兆交换机连接到主机 apolo、hermes、nix 和 artemis。在基层网络上，创建了由三台虚拟路由器组成的两个 VN。虚拟路由器的映像在一个库中，可通过网络文件系统（NFS）由基层网络中的所有机器访问。物理机器和 VM 都有操作系统 Debian GNU/Linux，内核版本为 2.6.32。

用于测试床的创建和代理操作的各 VM 的管理，使用 Libvirt 库［COU 10］。它为各种虚拟平台（包括 Xen）的监测和控制提供一个 API。在用于 VN 自管理的系统中使用 Libvirt，使之独立于采用的虚拟化技术。

各代理运行在网络核心的物理机器上：zeus、atlas、dionisio 和 cronos。使用 Ginkgo 平台［GIN 08］实现各代理。它支持轻量和可移植代理的创建，这有助于它们在异构环境中的部署。Ginkgo 是一个框架，它有架构的基本构造块。

多代理系统必须实施各 VN 的灾难恢复。由于虚拟路由器、物理节点，甚至物

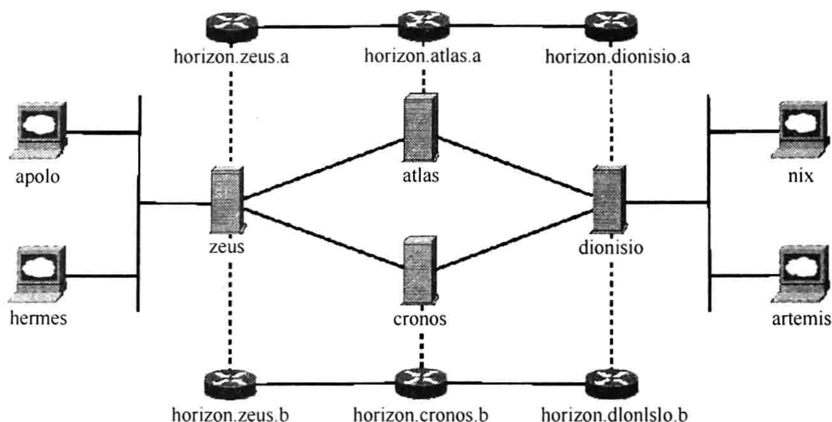


图 6.19　建立的测试床，确认虚拟网络自管理的多代理系统

理链路中的问题而发生故障。在第一种情形中，负责虚拟路由器的代理必须诊断故障和将故障通知其他代理。在其他情形中，该代理也可停止通信，因此各邻居必须诊断故障。

实现了用于 VN 自愈的一个自治控制环。该环受到动态规划器的控制，并由四种行为组成：Monitor、Analyze、Plan 和 Execute。各代理以这个顺序定期地实施这些行为。它们如下描述：

1） Monitor（监测）：从虚拟和物理节点收集数据，并馈入 KB。

2） Analyze（分析）：在虚拟路由器或物理节点内邻居关系上实施故障诊断。

3） Plan（规划）：基于基层节点的使用情况，计算它的成本。有多个虚拟路由器节点运行于其上的物理节点具有最高成本。它也将这个信息传播到其他代理。

4） Execute（执行）：验证邻居关系上所有代理是否已经发送它们的信息。如果是的话，则物理节点中具有最低成本的代理恢复失效的虚拟路由器。

在一台主机机器上创建一个特殊代理。这个代理是邻居关系的成员，并接收其他代理的信息。它执行一个行为来创建和更新一个图形界面。这个代理对形成这样一个架构原型是有贡献的，即由具有不同能力和认知等级的混合代理组成。在这种情形中，它在基层上实施拓扑的可视化和 VN 的映射，并以图表形式显示由其他代理的 Monitor 行为收集的信息。图形界面如图 6.20 所示。

6.8.2　试验结果

各试验的目的是验证 VN 自管理的架构，并在存在物理单元故障情况下，测试恢复虚拟路由器的不同方法。可依据下式，设置从一个 VN 的恢复时间 T_r：

$$T_r = T_d + T_p + T_i + T_c \tag{6.1}$$

式中，T_d是诊断故障的时间；T_p是花费在规划动作上的时间，涉及在各代理之间交

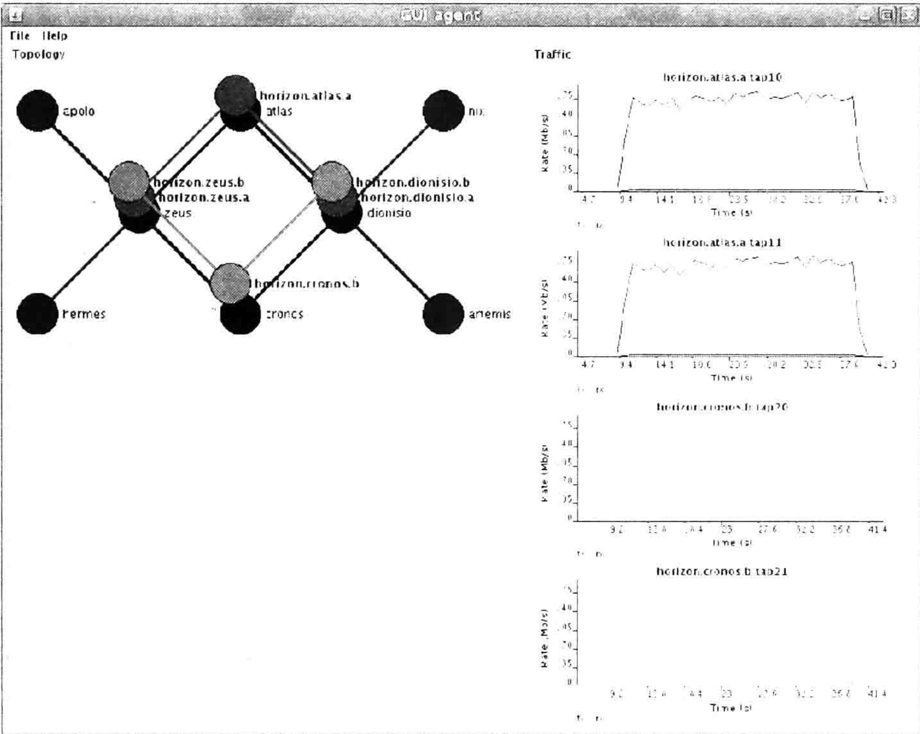


图 6.20　多代理系统的图形界面

换信息；T_i是 VM 的实例化时间；T_c是路由协议的收敛时间。

研究了恢复虚拟路由器的两种方式。第一种方式从库中的映射文件创建一个 VM，启动寄居操作系统。第二种方式从备份内存文件中（也在库中）恢复机器。在这种情形中，启动寄居操作系统就不需要时间，且路由协议的收敛时间就较小。

采用 VN 中的静态路由和动态路由实施各执行。静态路由器是人工配置在虚拟路由器中的。对于动态路由器，使用 Quagga 路由套件［ISH 13］，运行开放最短路径优先（OSPF）算法。

第一个试验是在没有多代理系统的条件下在测试床中实施的。实施了带有静态路由的一台虚拟路由器的迁移，同时一条 UDP 流以 500Kbit/s 恒定速率从 apolo 到 nix 传输通过 VN。采用 Iperf 工具产生流量。图 6.21 中的曲线给出流速率；一个零值指明网络中的丢失。所有迁移将虚拟路由器 horizon. cronos. b 的映射从 cronos 改变到 atlas，并在接近 8s 时开始。

在第一次运行中，图 6.21 中的曲线“创建”，在物理路由器 cronos 处销毁虚拟路由器，并在物理路由器 atlas 处重新创建。在这种情形中，恢复需要 27s，主要是由于启动操作系统的时间导致的。在第二次运行中，图 6.21 中的曲线“恢复”，

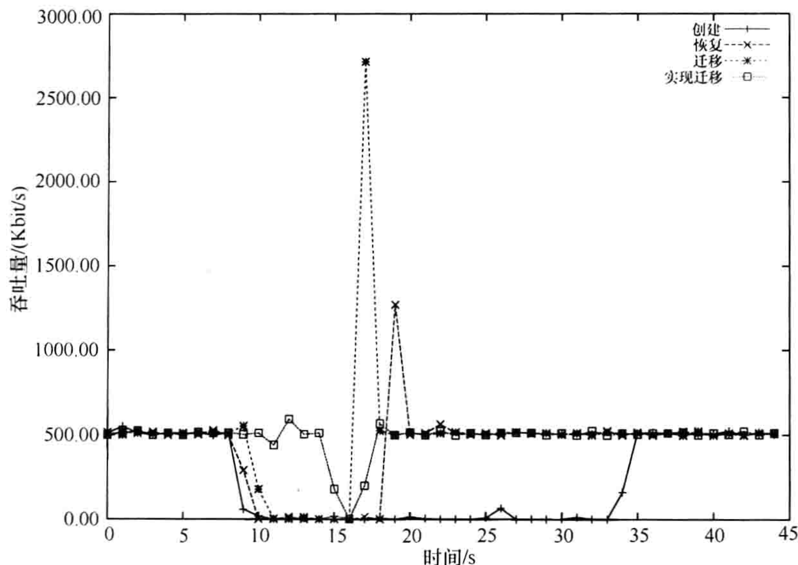


图 6.21 没有多代理系统的各试验

虚拟路由器的状态被存储在网络文件库中，在物理路由器 cronos 处停止虚拟路由器，保存在映像库中，并在物理路由器 atlas 处恢复。即使采用内存节省花费较多时间，该过程仅用去在第一个试验中所用时间的 45%。第三次运行使用 KVM 提供的迁移服务。在基本迁移中，在第三次运行中执行，图 6.21 中的曲线“迁移”，停止 VM，内存在网络上从源复制到目的地，之后在目的地处恢复该机器。这种状况类似于第二次运行，例外是复制直接从物理路由器 cronos 传递到物理路由器 atlas，而不是通过一个中间机器，且这得到的时间是第一次运行中所用时间的 37%。在这两种情况中，当机器恢复时，网络中存在一个峰值。发生这种情况是因为在保存 VM 内存时的时刻，在缓存中存在报文，这些报文被添加到 VM 被重新激活的时刻到达的那些报文。第四次运行，图 6.21 中的曲线“实现迁移”，是采用实况迁移实施的，其中在 VM 运行时将内存从源复制到目的地，且仅有此时，即控制被传递时，没有更多的修改页。采用这种方式，迁移一台虚拟路由器花费较少时间，原因是传输仅中断最短时长，相比第一次运行，时间要少 15%。

图 6.22 中的曲线给出分布式管理系统试验的结果，UDP 流量采用 Iperf 工具产生。图 6.22a 给出采用静态路由的执行情况，图 6.22b 采用动态路由。在这幅图中，比较了从映像文件创建虚拟路由器的方法以及从备份内存文件中恢复 VM 的方法。在所有运行中，一条以 20Mbit/s 恒定速率从 apolo 通过虚拟网络 B 发送到 nix，且各条曲线画出随时间变化的流速率。

在 10s 之后，物理路由器 cronos 从网络断开。zeus、atlas 和 dionisio 中的各代理诊断 cronos 的故障，原因是它们没有接收到 cronos 的 KB 传播。在这个试验中，

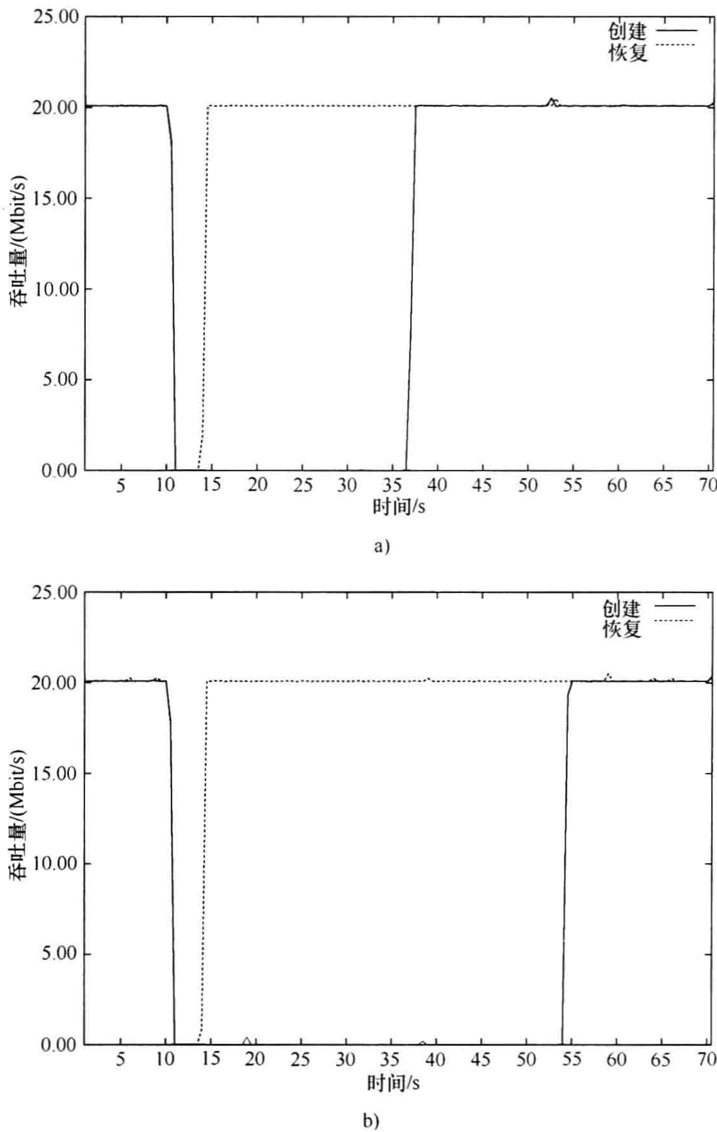


图 6.22 采用多代理系统和 Iperf 的各试验
a）静态路由 b）动态路由

各代理每隔 0.5s 实施 Monitor 行为和 KB 传播，有关一个物理节点的信息在 1s 以上就过期了。因此，故障诊断的时间变化范围在 1～1.5s 之间。之后，代理进入 Plan（规划）行为，此时它计算物理节点成本，这将在下一跳中发送到其他代理。Execute（执行）行为等待所有代理发送它们的物理节点成本。zeus 和 dionisio 的成本较高，原因是它们运行两个虚拟路由器。atlas 中的代理恢复 horizon. cronos. b；atlas 从映像中创建 VM，或从备份内存中恢复。

实施了在VN上安全复制协议（SCP）应用的另一项试验，结果如图6.23所示。SCP产生传输控制协议（TCP）流量。一个1GB文件使用VN B从hermes传输到artemis。对于每个场景，实施10次运行，采用95%的置信水平推导置信区间。所有执行都是在类似网络条件下实施的。

在这个试验中，出于比较目的，在没有代理作为基础的情况下实施各次执行。首先，测量了通过路径A（zeus→cronos→dionisio）和路径B（zeus→atlas→dionisio）发送文件的时间。如图6.23所示，路径A和路径B的流速率是不同的。因此，在启动SCP传输之后10s时，实施从cronos到atlas的实况迁移。这被用作比较的一个基础，即下限结果，因为实况迁移具有最低的下限时间。接下来，执行带有代理系统的试验，来实施VN的恢复。在这些场景中，也是在10s之后在物理节点cronos中发生一次故障。同样，多代理系统选择atlas恢复horizon. cronos. b。实施带有故障的四个场景，组合路由方案（动态的和静态的）和虚拟路由器恢复方法，或者从映像创建VM，或者从备份内存恢复VM。

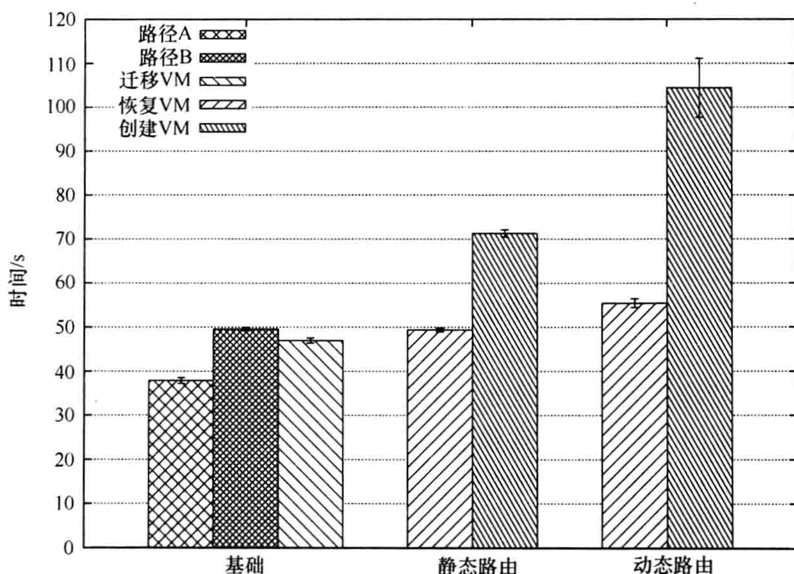


图6.23 采用多代理系统和SCP的各项试验

取决于要被恢复的虚拟路由器方式，在VN的恢复时间方面存在相当差异。当使用动态路由时，由于路由算法的收敛时间，差异是较大的。在这种情形中，通过创建VM的恢复花费43.6s，在采用多代理系统的Iperf试验中，有静态路由的方法用时26.4s。当与静态和动态场景中的下限值比较时，完成SCP传输的时间平均分别要高52%和122%。给定路由协议收敛时间，在动态路由场景中各VM的创建方法有一个较大的方差。

在VM状态从备份内存恢复的方法中，路由类型具有较低的影响。在静态和动

态场景中，在采用多代理系统的 Iperf 试验中，恢复时间大约为 3.5s。当与静态和动态场景中的下限值比较时，对 SCP 传输时间的影响分别大约为 5% 和 18%。

结果表明，从内存备份恢复 VM 的方法，特别当处理虚拟路由器时，会降低路由协议的收敛时间。虽然这种方法是比较快速的，节省传输和存储，但备份内存会导致对网络性能的严重影响。这种备份可发生在网络空闲的时刻，这导致对其操作的较少影响。也可使用分布式存储和内存气球法来降低这个额外负担。

6.9 小结

Horizon 项目提出一种新的系统或平面 PP，它支持网络中各自治控制环的协作。作为对变化语境的响应和符合高级目标与策略，PP 监管和集成网络控制门（doors）的行为。本章给出 Horizon 项目中 PP 的初始设计，定义了其功能，各组件操作背后的需求和概念。PP 寄居几个 APS，且它涉及一个或多个 PA 和一个动态 KB，KB 由一个数据模型和本体集合以及合适的映射逻辑组成。

一个 PA 是 Horizon 项目架构的一个功能实体，处理域间管理任务，如各管理域的联邦、协商、监管和分发。各 APS 有两个主要构造块。第一个是动态规划器，它用作一个自治的基于策略的分发器。动态规划器创建和销毁行为，并实现各 PA 和核心管理功能间的接口。第二个构造块代表一种行为，有两项功能。第一项功能，由核心行为实施，实现引导任务。第二项功能，由 PA 行为实现，作为各 APS 与 APS 所编排的各 PA 通信的一个代理。

另外，给出来自一个 APS 原型的一些结果，基于由 Ginkgo 平台提供的多代理来实现各 PA。

一个 APS 管理各 VN 的思路也可应用到一个云环境。图 6.24 形象地展示了在这个方面探索的一些思路。这幅图给出一个 VN，它的组成为：5 个 VM［它们在一个云环境（CloudRequest 的实例）中运行任务］构成的一个组和寄居在物理机器［连接到网络基层（VM-Server-Park）］上的各 VM 构成的组。网络管理由各 PA 完成，它们不断地监测资源。每种资源，无论物理的还是虚拟的，都至少有一个 PA，负责动作的监测和执行。各 PA 可被分组为域（邻居关系）。该图给出联合动作的 PA 的两个邻居关系。一个邻居关系由监测 VN（CloudRequest 的实例）的各 PA 组成，它们从另一个邻居关系中的各 PA 请求动作，后者监测网络基层（VM-Server-Park）。动作的一个例子可以是增加 VN 带宽的一条请求。图 6.24也形象地给出其他类型的 PA，辅助云的任务管理，如电源管理以及基于 SLA 的管理。

最后，给出一个自管理系统原型，其中通过使用一个多代理系统，将自治网络的概念应用到一个虚拟化环境。在要求自管理失效（自愈）的场景中，评估自治自管理环境。使用这个基础设施来构建原型。

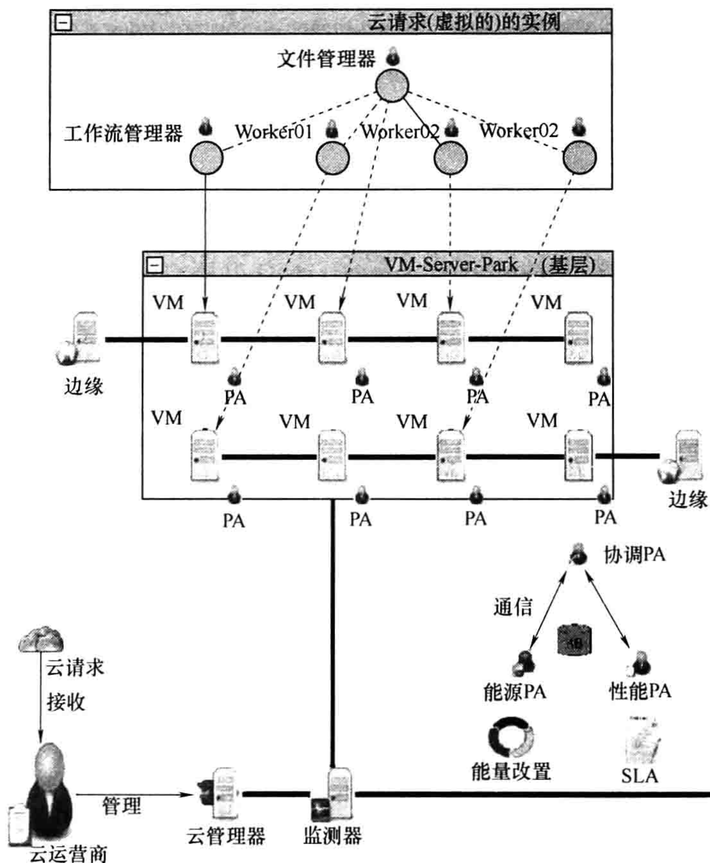


图6.24 虚拟网络管理的一种多代理模型

6.10 参考文献

[ANA 11] ANA P., "Autonomic network architecture", 2011, available at http://www.ana-project.org/.

[AND 05] ANDERSON T., PETERSON L., SHENKER S., *et al.*, "Overcoming the internet impasse through virtualization", *Computer*, vol. 38, pp. 34–41, 2005.

[BEL 05] BELLARD F., "Qemu, a fast and portable dynamic translator", *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC 05, USENIX Association, Berkeley, CA, pp. 41–46, 10–15 April 2005.

[BIO 11] BIONETS P., "Bio-inspired service evolution for the pervasive age", 2011, available at http://www.bionets.eu/.

[BUL 08a] BULLOT T., KHATOUN R., HUGUES L., *et al.*, Merghem-Boulahia L., "A situatedness-based knowledge plane for autonomic networking", *International Journal of Network Management*, vol. 18, pp. 171–193, 2008.

[BUL 08b] BULLOT G.P.T., GAÏTI D., ZIMMERMANN H., "A piloting plane for controlling wireless devices", *Telecommunication Systems*, vol. 39, pp. 195–203, 2008.

[CAS 11] CASCADAS P., "Component-ware for autonomic situation-aware communications, and dynamically adaptable services", 2011, available at http://acetoolkit.sourceforge.net/cascadas/.

[CHE 06] CHENG Y., FARHA R., KIM M.S., *et al.*, "A generic architecture for autonomic service and network management", *Computer Communications*, vol. 29, pp. 3691–3709, 2006.

[CLA 03] CLARK D.D., PARTRIDGE C., RAMMING J.C., *et al.*, "A knowledge plane for the internet", *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM 03, Karlsruhe, Germany, pp. 3–10, 25–29 August 2003.

[COU 10] COULSON D., BERRANGE D., VEILLARD D., *et al.*, "Libvirt 0.7.5: application development guide", 2010, available at http://libvirt.org/guide/pdf/Application_Development_Guide.pdf (accessed in January 2013).

[DOB 06] DOBSON S., DENAZIS S., FERNÁNDEZ A., *et al.*, "A survey of autonomic communications", *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, pp. 223–259, 2006.

[FAJ 10] FAJJARI I., AYARI M., PUJOLLE G., "VN-SLA: a virtual network specification schema for virtual network provisioning", *International Conference on Networking*, Paris, France, pp. 337–342, 11–16 April 2010.

[FEA 07] FEAMSTER N., GAO L., REXFORD J., "How to lease the Internet in your spare time", *SIGCOMM Computer Communication Review*, vol. 37, pp. 61–64, 2007.

[GAÏ 96] GAÏTI D., PUJOLLE G., "Performance management issues in ATM networks: traffic and congestion control", *IEEE/ACM Transactions on Networking*, vol. 4, Issue 2, pp. 249–257, 1996.

[GAÏ 06] GAÏTI D., PUJOLLE G., SALAUN M., *et al.*, "Autonomous network equipments", *Autonomic Communication*, LNCS – Lecture Notes on Computer Science, vol. 3854, pp. 177–185, 2006.

[GIN 08] GINKGO NETWORKS, "Ginkgo distributed network piloting system", *White Paper*, 2008, available at http://www.ginkgo-networks.com/IMG/pdf/WP_Ginkgo_DNPS_v1_1.pdf (accessed in January 2013).

[GRE 05] GREENBERG A., HJALMTYSSON G., MALTZ D.A., *et al.*, Refactoring network control and management: a case for the 4D architecture, Technical report, 2005.

[HAG 11] HAGGLE P., "A content-centric network architecture for opportunistic communication", 2011, available at http://code.google.com/p/haggle/.

[HOU 08] HOUIDI I., LOUATI W., ZEGHLACHE D., "A distributed and autonomic virtual network mapping framework", *4th International Conference on Autonomic and Autonomous Systems*, ICAS 08, pp. 241–247, March 2008.

[HOU 10] HOUIDI I., LOUATI W., ZEGHLACHE D., *et al.*, "Adaptive virtual network provisioning", *Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA 10, ACM, New York, pp. 41–48, 2010.

[IBM 06] IBM, "An architectural blueprint for autonomic computing", *Autonomic Computing White Paper*, 4th ed., June 2006.

[ISH 13] ISHIGURO K., "Quagga: a routing software package for TCP/IP networks", 2006, available at http://www.quagga.net/docs/quagga.pdf (accessed in January 2013).

[KEP 03] KEPHART J.O., CHESS D.M., "The vision of autonomic computing", *Computer*, vol. 36, pp. 41–50, 2003.

[KIV 07] KIVITY A., KAMAY Y., LAOR D., *et al.*, "KVM: the linux virtual machine monitor", *Proceedings of the Linux Symposium*, vol. 1, pp. 225–230, 2007.

[MAR 10] MARQUEZAN C.C., GRANVILLE L.Z., NUNZI G., *et al.*, "Distributed autonomic resource management for network virtualization", *IEEE/IFIP Network Operations and Management Symposium*, NOMS 10, Osaka, Japan, pp. 463–470, April 2010.

[NIE 05] NIEBERT N., "Ambient networks: a framework for mobile network cooperation", *Proceedings of the 1st ACM Workshop on Dynamic Interconnection of Networks*, DIN 05, ACM, New York, pp. 2–6, 2005.

[PFA 09] PFAFF B., PETTIT J., AMIDON K., *et al.*, "Extending networking into the virtualization layer", *Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, New York, October 2009.

[SEN 11] SENNA C.R., BATISTA D.M., SOARES M.A. JR, *et al.*, "Experiments with a self-management system for virtual networks", *Proceedings of the 2nd Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF 2011)- XXIX Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos*, SBRC 11, Brazilian Computer Society, Brazil, 2011.

[SOA 12] SOARES M.A. JR, MADEIRA E.R.M., "A multi-agent architecture for autonomic management of virtual networks", *Proceedings of the 4th IEEE/IFIP International Workshop on Management of the Future Internet*, ManFi 12, IEEE Computer Society, Maui, USA, 16 April 2012.

[STR 06] STRASSNER J., AGOULMINE N., LEHTIHET E., "Focale: a novel autonomic networking architecture", *Proceedings of the Latin American Autonomic Computing Symposium*, LAACS 06, Campo Grande, Brazil, 18–19 July 2006.

[TSE 09] TSELENTIS G., DOMINGUE J., GALIS A., *et al.*, *Towards the Future Internet, A European Research Perspective*, IOS Press BV, 2009.

[TUR 05] TURNER J., TAYLOR D., "Diversifying the internet", *In Proceedings of the IEEE Global Telecommunications Conference, GLOBECOM*, IEEE, St. Louis, USA, pp. 755–760, December 2005.

[YU 08] YU M., YI Y., REXFORD J., *et al.*, "Rethinking virtual network embedding: substrate support for path splitting and migration", *SIGCOMM Computer Communication Review*, vol. 38, pp. 17–29, 2008.

[ZHU 06] ZHU Y., AMMAR M., "Algorithms for assigning substrate network resources to virtual network components", *INFOCOM 2006, Proceedings of the 25th IEEE International Conference on Computer Communications*, Barcelona, Spain, April 2006.

第7章　管理和控制：共置视图

网络虚拟化技术支持在唯一的物理基层之上运行多个虚拟网络。通过使用一个控制和管理实体取得这项功能，该实体复用硬件访问，并向被虚拟系统提供资源的逻辑分片。一个被虚拟化的联网系统的主要原语是：虚拟网络的创建和销毁，虚拟节点的迁移以及虚拟网络在物理基层上的映射。

在 Xen 和 OpenFlow 的标准版本中，所有上述原语都是手工运行的，这意味着扩展性和管理问题。因此，提出和开发了一个引导平面，该平面能够自治地执行这些原语。也开发了一个知识平面，它与引导平面一起工作，检测何时虚拟网络不再按照期望那样工作，则结果就要求对其属性的修改（通过执行原语）。知识平面监测虚拟路由器，得到它们的使用情况概要，并通过使用预测机制，先验地检测虚拟网络配置中更新的必要性。知识平面存储与每个虚拟网元有关的信息，支持管理决策和先验的网络维护。因为扩展性问题，知识平面被分布在不同节点中。因此，每个节点保持知识平面的一个部分视图，该视图限制在其邻居关系和周边环境。这个部分视图被称作一个节点的共置视图。知识平面的主要挑战是采取决策的时间调度和网元信息更新。定义和选择必须存储于知识平面上的信息，还有这个信息将被存储在哪些节点上。最后，基于性能和服务质量（QoS）度量元，为每种类型的信息定义更新频率。由于扩展性，仅处理共置视图信息而不是整个网络信息。

开发了一个分布式机制集合，来检测和修正虚拟网络异常。各机制识别和存储网络状态变化，也预测可变的演化情况。所提供机制良好地适合于 Xen 和 OpenFlow 平台，但这里仅描述 Xen 平台的机制。

提出的第一种方法基于一种动态分配系统的开发，该系统分析一台虚拟路由器的资源利用率概要，并基于 QoS 度量元和服务水平协议（SLA）提供资源的公平共享。提出的第二种方法焦点是监测和预测技术，这些技术监测环境，向知识平面提供合适的和更新的信息，也检测路由器的不当行为。这两种方法一起构成检测更新需要以及当违反 SLA 时（也可能触发更新告警）的一个框架。

本章组织如下。7.1 节描述抽取和存储虚拟网络概要的机制。同时，依据得到的概要和定义好的 SLA（基于模糊逻辑），描述所实现的一个 QoS 控制器。7.2 节详细解释监测套件和预测器，它们用来检测从各节点得到的许多参数中变化的需要。7.3 节对本章做出结论。

7.1 动态 SLA 控制器

本节给出评估一台虚拟路由器环境上更新需要的系统。这个系统动态地控制一个虚拟化网络环境的 SLA，并提供其 QoS 保障。

7.1.1 有关虚拟网络 QoS 的背景知识

在参考文献中找到的人们提出的动态分配机制主要将焦点放在服务器合并上，没有很好地涵盖虚拟路由器资源分配。Sandpiper [WOO 07] 是监测一个数据中心中虚拟机的一个系统，并将虚拟机迁移到不同物理服务器，目的是取得一种分布式虚拟机配置，这种配置可最大化性能并降低资源的滥用。为避免诸如拒绝服务（DoS）攻击等不当行为，通过使用时间序列，Sandpiper 分析了每台虚拟机的资源使用概要。每台虚拟服务器的资源利用率由一个体量度量元 Vol 定义，表示为

$$\text{Vol} = \frac{1}{1-\text{cpu}} * \frac{1}{1-\text{mem}} * \frac{1}{1-\text{net}} \tag{7.1}$$

式中，cpu 是处理器利用率百分比；mem 意指内存利用率；net 表示网络利用率。

Meng 等 [MEN 10] 提出在一个由物理服务器组成的网格中提供最佳虚拟化服务器分配的算法，目的是改进网络扩展性并优化通信链路中的带宽利用率。为减少相互交换数据的服务器之间的距离，在物理服务器上实例化虚拟服务器。Menascé 等 [MEN 06] 将自治计算技术应用到控制虚拟机间的处理器共享。作者提供一种动态分配算法，通过仿真进行了验证。

Xu 等 [XU 08] 基于模糊逻辑提出在数据中心中优化资源分配的控制算法，同时采用不同负载执行虚拟化 Web 服务器中的性能测试。一个学习系统馈入模糊控制器，描述在不同负载下 Web 服务器行为。

有助于动态资源分配的一个重要工具是虚拟机迁移，这在 Sandpiper [WOO 07] 中得到使用。迁移原语支持虚拟机从一台物理服务器迁移到另一台物理服务器，这支持防御性维护和能量节省，是通过机器的重新组织和关闭欠利用（underutilized）的服务器得到的。不过，当在一项虚拟服务器应用中实施虚拟机迁移规程时，由于当机器处于悬挂时的时段过程中的报文丢失，该迁移规程代表一项巨大挑战。Wang 等 [WAN 07] 提出没有报文丢失的一种实况迁移机制，Pisa 等 [PIS 10] 在 Xen 架构中实现了这项建议。

在 Xen 平台中动态资源分配和控制是一项挑战，这是因为输入/输出（I/O）虚拟化技术仍然是原生的，没有隔离，这使恶意虚拟路由器可影响同一台物理路由器中其他虚拟路由器的性能。因此，XNetMon [FER 10a] 提出一种路由流量的安全控制系统，它基于数据和控制平面隔离的方法，管理由虚拟路由器对 I/O 资源的使用情况。

Keller 等［KEL 09］分析了虚拟化环境中的 QoS 需求，并提出一个授权模型来描述和保障虚拟路由器中的 SLA。作者论证了授权机制的优势，并提出两种主要的实现模型。第一种模型基于网络参数监测，而第二种基于使用信任平台和密码学密钥，提供防误用属性。

本章给出针对虚拟网络的一种基于 SLA 的动态控制器。这种控制基于路由器概要的生成和进一步分析、SLA 违规检测和不当行为虚拟路由器的实时惩罚。所提供系统监测与每个虚拟网络相关联的负载值，并产生使用概要的实际估计。这些概要确保降低过载概率的资源分配。负载函数基于 Sandpiper 建议，但也包括其他重要参数，如系统鲁棒性、运算处理器温度和其他参数（如果需要的话，可以添加）。所提供模糊控制系统目标是提供一种简便的加权参数配置。模糊逻辑映射管理员战略，并监测 SLA 违规的路由器。此外，网络管理员可容易地插入新的规则和动作战略。所提供系统支持数据/控制平面隔离，并作为一项后果，是完全与 XNetMon 兼容的。

从原型得到的结果给出控制系统的行为、所产生的概要和战略机制。控制器的执行仅消耗一些中央处理单元（CPU）周期，这支持并行地监测许多台虚拟路由器。

7.1.2 建议的模糊控制系统

建议的模糊控制系统监测和确保虚拟化网络环境中的 SLA。关键思路依赖于每个虚拟网络使用概要的生成和分析、SLA 违规的监测和不当行为虚拟路由器的实时惩罚。SLA 违规检测技术考虑违规的严重程度，并将严重程度映射到一个惩罚等级，将之应用到每台行为不当的虚拟路由器。系统负载也影响惩罚等级。因此，路由器和控制域的全局状态也通过使用一个模糊控制器来刻画，其中当计算系统负载时，考虑了处理器、内存、网络和鲁棒性（存在冗余性和防失效机制）。依据控制器的输出，动态地修改惩罚战略和系统对违规动作的容忍度，是可能的。

所建议系统是由控制器代理组成的一个分布式管理系统。每个代理控制一个物理路由器集合及其关联的虚拟路由器，如图 7.1 所示。每台物理路由器有一个控制域，其中一个控制和监测守护进程监测物理资源的分配，实时地验证 SLA 的符合性，并生成每台虚拟路由器的使用概要。由 5 个模块组成控制器代理。战略和策略模块（SPM）持有可在其控制下物理路由器上应用的管理战略，并更新可应用在每个控制和监测守护进程的当前战略。服务水平模块（SLM）保持一个数据库，它将 SLA 与每台虚拟路由器关联。知识库模块存储历史、使用概要和已经发生的违规的描述。这个模块估计未来网络迁移，检测更新的需要，也协商契约，这使契约适合每台虚拟路由器的真实概要。执行器模块在控制和监测守护进程内执行，并检索每台虚拟路由器的概要和统计信息。控制器也有一个通信模块，支持通过安全信道在其他控制器间交换信息。若有必要，控制器可使用那些信道协商执行域的变化，并

协商虚拟单元的迁移。

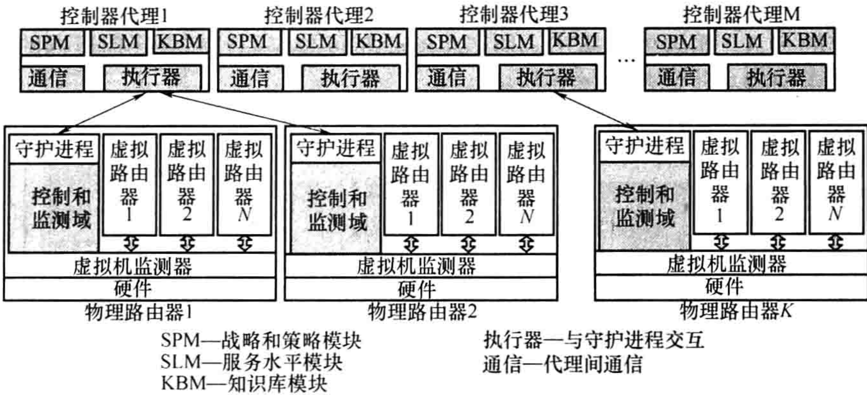


图 7.1　控制系统架构：分布式控制器代理的执行器模块与运行在物理路由器上的监测和控制守护进程交互

所述系统有三个主要机制。第一个机制是概要生成机制，它提供利用率统计信息和 SLA 违规检测。这个信息被存储在 KBM 中，目的是重新协商达成契约的 SLA 中的可能的误配置。第二个机制是系统负载估计器，它给出组合多个资源状态在［0,1］区间内一个估计的输出。第三个机制是自适应惩罚机制。基于系统负载和使用概要，该机制使用一个模糊控制器，它输出一个惩罚等级，正比于系统整个状态。例如，如果系统给出一个低的负载，则一个中等违规（如超过 20% 的 SLA）生成一次小的惩罚（将违规 SLA 机器的资源利用率降低 2%）。另外，如果系统过载，在没有可用资源的条件下，即使一个小的违规也会被严厉地惩罚。

1. 生成路由器概要

每台虚拟路由器的使用概要代表每台虚拟路由器的资源消耗模式。使用各概要检测规则违规，估计资源消耗，并预测未来需要。通过捕获随时间变化的内存、处理器和网络利用率，生成概要，存储这些变量的最近过去和长远过去的行为。有两个不同尺寸的滑动窗口，存储这两个关联的时间序列。在 Sandpiper［WOO 07］中使用基于概率密度函数的概要生成方法。使用最近的过去来检查 SLA，而使用长远的过去来预测未来行为和估计可能的未来需求。采用概率函数分析各行为。

执行 RIPv2 的一台给定虚拟路由器，它的一个处理器利用率的概率密度函数（PDF）如图 7.2a 所示。在这台特定路由器中控制和数据消息的交换，在长远过去窗口过程中 70% 的测量数据（由这个场景的过去 200 个测量数据组成）中，这产生了约 0.7% 的 CPU 处理器使用率。考虑到这个 PDF，得出结论，在不损失性能的情况下，聚合具有类似资源模式的一组虚拟路由器，并使它们共享同一物理核，是可能的。为验证 SLA，原型也提供累积分布函数（CDF）。灵活的 SLA 是可能的，

且可定义（例如）一台虚拟路由器在其执行时间的最大80%时间，可使用一项给定资源的高达0.7%。通过最近过去窗口的CDF，则可确定图7.2b中的路由器将满足给定的SLA。重要的是指出，所产生的分布可被应用到来自任何机器的任何路由器。给出的例子仅说明可得到哪种分布。

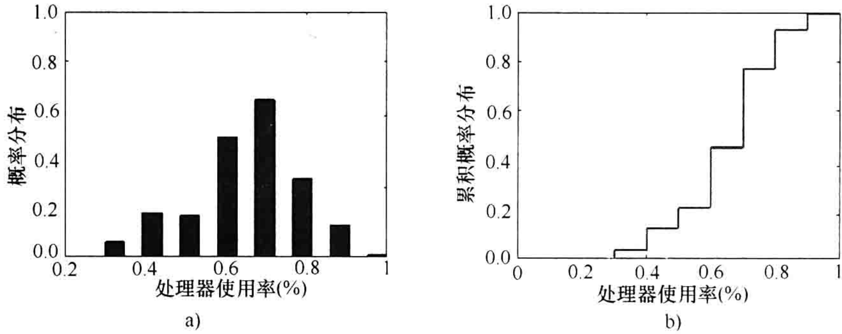


图7.2 运行RIPv2的一台虚拟路由器的处理器利用率概要
a）概率密度函数（PDF） b）累积分布函数（CDF）

2. 战略和策略模块

SPM存储当前发挥作用的战略，并将管理决策映射到动作和战略。使用模糊控制器［KEC 01］，由于其处理决策问题的能力，它给出不确定性参数和定性参数，如网络管理员或管理人员的战略。在模糊逻辑中，如前所述，一个给定元素属于一个给定集合，依据的是在区间［0，1］内其成员管理等级，其中$\mu_A(x)$：$X\rightarrow[0,1]$定义一个成员关系函数。采用带有Zadeh的［ZAD 00］AND和OR算子的Mamdani隐性方法与去模糊化的质心方法。模糊控制器具有小的计算复杂度，并可并行化推理过程，这增强了系统性能并降低了控制器的响应时间。

SPM支持不同的作用战略。每个战略由一个推理规则集、一个成员关系函数集［依据网络管理员的感知（如“高处理器利用率”和“低内存负载”）来映射输入参数］和一个约束输出的成员关系函数组成。有两种战略类型：系统负载战略和惩罚战略。在第5部分“负载策略”描述这两种方法及其相关的战略包。那些战略形式化一个计算行为，它反映网络管理员的意愿和战略。

3. 估计系统负载

系统负载是确定被管资源负载水平的一个测量数据。诸如处理器利用率、内存利用率、网络利用率、系统整体温度和系统鲁棒（指明存在磁盘和供电的冗余机制）等多个参数，可被分析来刻画系统状态。定义成员函数集合μ_{Proc}、μ_{Mem}、μ_{Net}、μ_{Temp}和μ_{Rob}，这与模糊化变量中的每种资源关联。参数的组合产生一个输出，定义为系统负载，被限制在区间［0,1］内，它定义系统的负载。这个值被用作另一个控制器的一个输入参数，与delta（是契约资源和资源之间的差）一起，来估计违反SLA的路由器的惩罚水平。图7.3给出代表系统负载控制器

的一个框图。

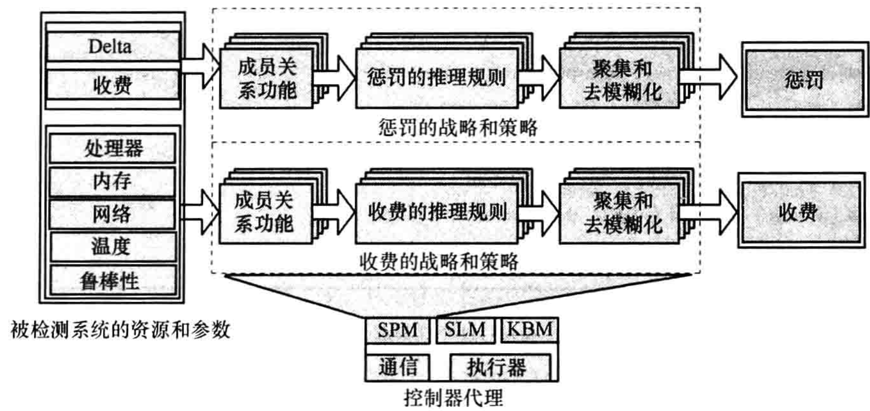


图 7.3 战略和策略模块

汇聚每台虚拟路由器的资源使用情况，产生控制器的输入。在图 7.4 中可看到评估处理器使用和系统温度的一种可能配置的例子。重要的是记住，依据每名管理员的需要和定性想法，可修改给出的曲线。低、中、高、冷、温和热是成员函数。在给定配置中，使用了三个成员函数来映射每种资源。成员函数的定义代表网络管理员定性决策的映射。多数规则可被定义为三角函数或梯形函数。例如，如果处理器使用较高，且总体系统温度较高，那么系统过载也较高。

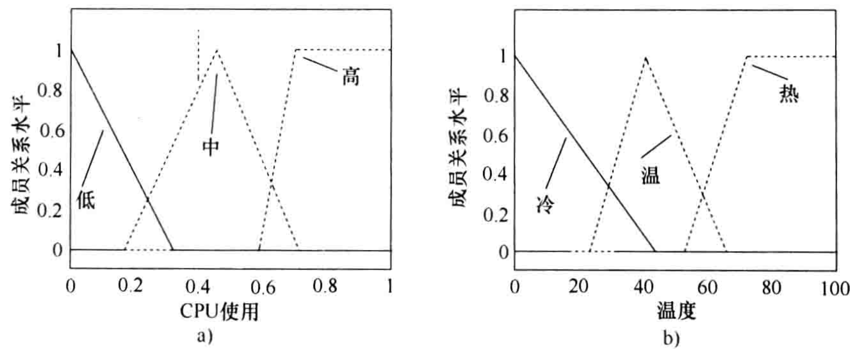


图 7.4 处理器和温度利用率的成员函数

a）处理器的成员函数 b）温度利用率的成员函数

4. 基于推理规则的战略

模糊控制器战略基于模糊系统的默认模糊推理规则。这些规则遵循 IF→THEN 模式，这代表当前的动作战略方案。定义为战略的规则集被称作一个战略包。战略包的一个例子依据契约 SLA 和当前资源使用情况的差（表示为 delta）与系统负载计算惩罚等级，这可从表 7.1 看到。

表 7.1　一个战略包一部分（piece）的例子

战略包
If **delta** (low) and **load** (low), then **punishment** (low)
If **delta** (average) and **load** (low), then **punishment** (low)
If **delta** (high) and **load** (low), then **punishment** (average)
If **delta** (low) and **load** (high), then **punishment** (average)
If **delta** (average) and **load** (high), then **punishment** (high)
If **delta** (high) and **load** (high), then **punishment** (high)

给出的战略包对应于一个网络管理员战略，它建立这样的规则，当系统轻载时，即使大量 SLA 违规也不被严厉地惩罚，原因是系统具有丰富的资源，且此刻将额外资源赋予违规路由器不会干扰其他路由器。但是，当系统过载时，网络管理员是比较严格的，即使轻的违规也要受到严厉惩罚。这些规则与成员函数一起工作，成员函数也可由网络管理员开发。新的规则和战略被方便地插入，且控制器代理必须将战略包输出到守护进程，守护进程必须使用所定义的战略。也可为在控制下的每种资源建立不同战略，增强控制器的灵活性。

5. 负载策略

推理规则提供表示每个推理规则相关等级的模糊值，之后这些值被映射到单个控制器输出，它是区间［0,1］中的一个值，这代表当前的系统负载。图 7.5 给出两个可能的负载策略：一个保守的负载策略和一个大胆的负载策略。取决于网络管理员概要，可动态地改变当前的域策略。

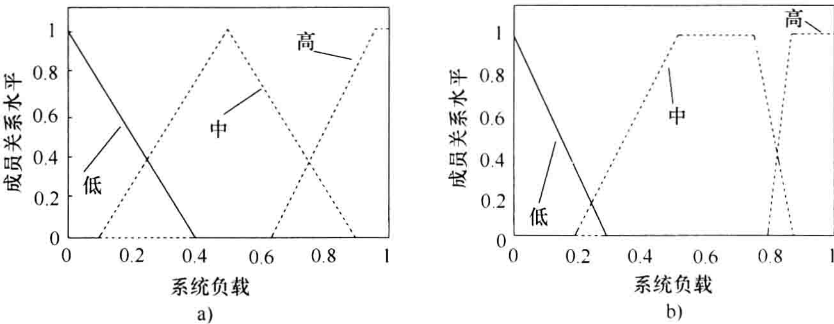


图 7.5　系统收费（charge）策略
a）保守策略　b）大胆策略

图 7.6 给出系统的决策平面，代表一个系统负载的惩罚等级和 delta，即 SLA 违规等级。当 delta 较高且系统过载时（图 7.6a），保守策略仅严厉地惩罚一个行为不当的虚拟路由器，而即使对于系统负载和 delta 的较小正的变化，大胆的策略也产生高等级的惩罚（图 7.6b）。

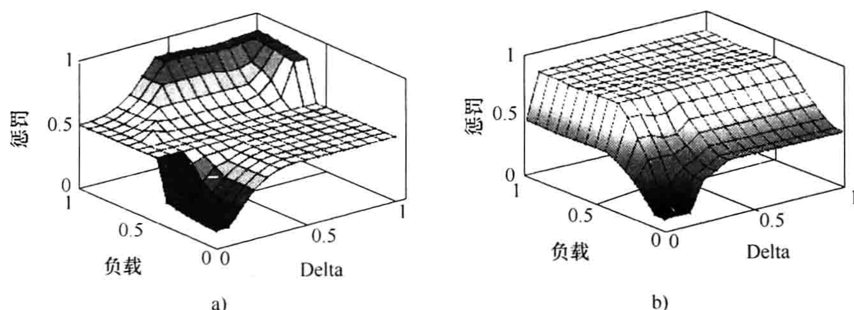


图 7.6　不同管理战略的决策表面

a）一个保守策略的决策表面　b）一个大胆策略的决策表面

6. 控制系统过载和 SLA

开发的系统生成利用率概要，评估概要是否对应于已建立的 SLA，计算系统负载的一个估计，并惩罚违反所提出规则的虚拟路由器。在每个域内执行的守护进程实施在每个给定采样间隔收集的参数，采样间隔可由网络管理员定义。这些参数被用来产生代表每种资源利用率行为的时间序列以及统计信息和分布，支持验证概要和与 SLA 的符合性。所有这些信息被发送到负责的控制器代理。守护进程验证每个虚拟路由器概要是否对应于协商过的 SLA。此外，它汇聚由每台路由器使用的资源，估计物理系统的总负载。如果一台路由器违反契约，则系统产生代表 delta 的一个值。之后系统使用这个 delta 值和系统负载来确定要被应用到不当行为路由器的合适惩罚等级。在 Xen 架构中，使用上界（cap）控制参数。上界约束每个虚拟单元可使用的 CPU 周期数。因此，改变上界值，可控制虚拟路由器的处理器资源的使用。可被使用的另外一个控制工具是流量控制（TC），这支持队列控制，如果虚拟路由器违反 SLA，则允许对每台虚拟路由器的吞吐量实施管理。

7.1.3　结果

开发的模糊控制系统是高效的和灵活的。开发了一组试验，证明由所提出的系统引入的效率和低额外负担。在一台物理机器上实施测试，这台机器有一个核心 i7 860处理器，处理器有 4 个真实核和 8GB DDR3 随机访问内存（RAM）。该机器配置有 hypervisor Xen 4.0。虚拟路由器实例化有 128MB RAM 内存，可访问一个虚拟核。虚拟路由器和控制域执行 2.6.32-5-amd64 内核的 Debian Lenny Linux。

控制系统的设计最小化了监测和控制守护进程的处理额外负担。为评估守护进程的处理器额外负担，实例化了一些虚拟路由器，并依据被检测虚拟路由器数，测量了控制域的平均处理器利用率。图 7.7 的结果给出依据被监测虚拟路由器数，在控制域中的处理器利用率。在这个配置中，在每秒收集并评估测量结果和决策。图中的各点代表每个配置的平均处理器利用率，置信区间为 95%。可观察到，处理

器利用率和被监测路由器数之间的关系近似为线性的，甚至在守护进程同时管理 8 台路由器的情形中，额外负载也是可接受的，且达到单核处理器利用率的 40%。

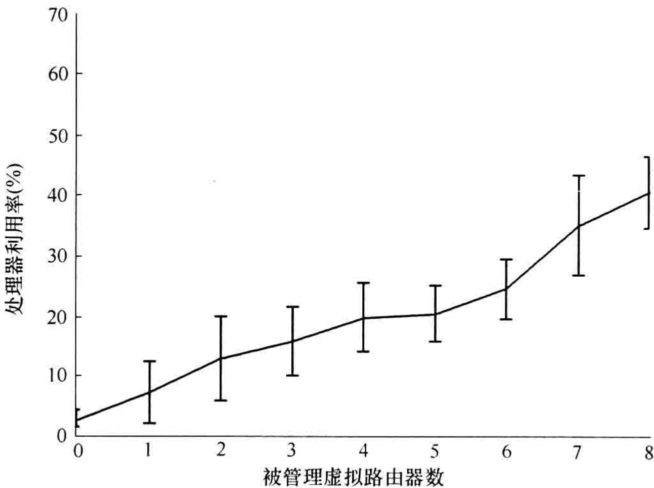


图 7.7　当被管理虚拟路由器数变化时控制域中的处理器利用率

可估计，与这个试验中所使用相同配置的一个控制域，通过仅将单个核专用于这项任务，就可同时管理多达 20 个虚拟路由器。因此，系统给出不错的性能，原因是它同时监测来自多个虚拟路由器的多个变量，且那个系统正在管理每台虚拟路由器的 SLA。

第二个试验评估控制器的效率和惩罚机制的影响。选择虚拟路由器中的一台路由器。这台被选中虚拟路由器的 SLA 定义，为执行报文转发，它可使用单核高达 85% 的处理器。接下来，创建要被这台路由器转发的一条报文流。当流被转发时，路由器违反 SLA，且控制系统通过上界机制规范处理器利用率。在试验中，当路由器已经正在违反 SLA 时，激活惩罚系统。

在所提出的场景中，定义三个背景环境。在第一个环境中，有一台被监测路由器和一台没有使用资源的虚拟路由器。所以，系统保持低的负载。在第二个环境中，有被监测的路由器和正在消耗中等量资源的 5 台虚拟路由器。在这个场景中，负载是中等的。第三个环节有被监测的虚拟路由器和 7 个额外的虚拟路由器，都在使用几乎所有可用的资源。在这种情形中，系统负载被刻画为高负载。因此，场景特征由低、中和高负载输出。得到这些值，考虑到系统上的一组成员关系函数和推理规则。在每种配置中，取决于背景环境，系统产生不同的负载输出。从被测的所有环节中选择三个特定环境，来展示对每种可能的系统负载输出时系统的不同行为。图 7.8 中给出的结果，展示出系统收敛到可确保路由器 SLA。取决于系统的负载等级，惩罚等级是变化的。观察到，在低负载环境中，系统用时 40s 达到契约 SLA。当存在丰富的资源时，SLA 违规不会伤害其他路由器。在中等负载场景中控

制器的使用诱发惩罚等级的提高，结果，在 15s 内资源滥用得到控制。在高负载场景中，惩罚是严厉的，系统在 5s 内限制资源使用。因此，所提供控制器是高效的，并满足已建立的需求，在低负载场景中以一种保守方式起作用，在关键情况下以大胆果断的方式起作用。

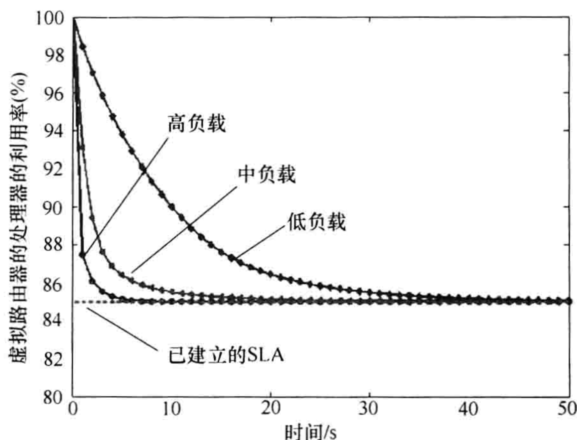


图 7.8　不同系统负载下的系统稳定性：当系统负载为高负载时，由于更加严厉的惩罚，比较快速地收敛到契约 SLA

第三个试验评估暂时不当行为的控制器效率，此时一台给定虚拟路由器在一个给定时长违反 SLA，之后它开始遵守契约。虚拟路由器以一个高报文速率转发报文，并使用其处理器的 100%。在 60s 后，路由器以一个较低速率转发报文，并消耗其处理器的 80%，不再违反 SLA。在这个结果中，系统负载是可控制的，并被分类为中等负载。图 7.9 表明，在没有控制器的条件下，虚拟路由器可使用它希望

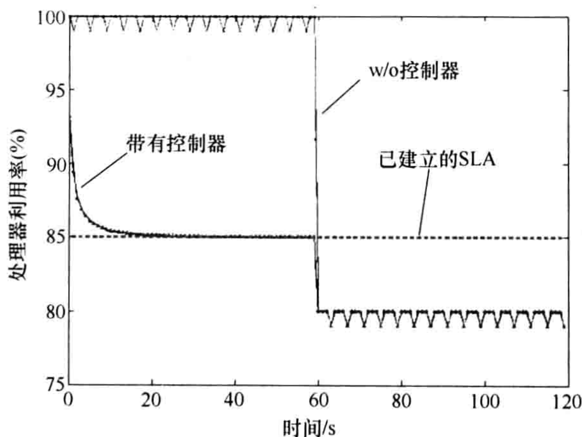


图 7.9　一台虚拟路由器的处理器利用率：该路由器在一个给定时长违反一条 SLA，具有中等系统负载

使用的情况，可能会伤害其他路由器。当控制器起作用时，虚拟路由器受到一个递减的上界约束，直到虚拟机器的处理消耗收敛，符合 SLA 时为止。

7.2 局部信息的更新预测机制

为虚拟网络系统的自治管理提出一种异常检测（ADAGA），为在虚拟网络环境中收集和分析数据提供机制。ADAGA 管理器观测网络中的被监测系统，如服务器和路由器，无论物理的还是虚拟的。所提供系统的主要目标是发送告警，报告网络实体发生的可能异常。这项研究将一次异常行为刻画为观测中短期的变化，它与过去的测量数据不一致。

考虑到序列的过去历史，ADAGA 使用时间序列预测实际值，并与新的观测值比较。ADAGA 考虑所有时间序列都初始化为 0。对于这一点，每个序列的初始值降低它的所有观察值，这允许在初始化过程中的零误差，对未来预测的初始条件没有影响。正确的预测器初始化是一项重要的配置，原因是它影响整个系统的性能[BRU 00]。

预测器分析基于当预测器参数变化时的假阳性和假阴性值。这个报告分析被监测系统的行为，描述异常状况，并给出当发生一次异常状况时对发出告警的影响。各试验是在一台真实路由器上实施的，人工产生异常来仿真路由器中的一次过载。结果表明，ADAGA 系统检测到给出平均假阳性和假阴性率的异常。

7.2.1 节讨论与自治管理和异常检测有关的研究。7.2.2 节描述所开发的系统及其模块。7.2.3 节给出试验的测试场景，是采用实现的原型实施测试，并得到结果。最后，7.3 节做出结论并给出未来工作的方向。

7.2.1 异常检测系统的背景

基于异常检测的虚拟网络监测和管理是一个没有很好描述过的专题。异常检测技术普遍用于安全（如入侵检测系统［PAT 07］）中，也被用于网络管理的自治系统，由一次异常检测和发出一次告警来触发。异常可分为三种类型［BAR 01］：网络操作运行中的异常，由网络设备故障或配置改变组成；由一次瞬态群集导致的异常，当许多用户同时请求特定位置信息（如一个操作系统新版本或一个病毒视频的分发）时通常发生这种异常；由网络滥用导致的异常（如 DoS 攻击和端口扫描）。所提出的系统考虑了所有这些异常，原因是这些对自治虚拟网络管理的用户满意度具有重大影响。

Brutlag［BRU 00］对计算机网络中的异常检测使用时间序列和预测机制。作者将焦点放在分析由一台路由器产生报警的网络流量异常。给出预测器的单参数配置结果，预测器被初始化为零，在序列中没有任何数据预处理。在 ADAGA 中，为在启动时得到零误差，对接收到的数据进行预处理。此外，ADAGA 和 Brutlag 的系

统之间的主要差异是多维分析，其中将时间序列的计算应用到不同网络特征，如内存和处理器利用率、网络流量和系统负载。在虚拟化场景中，由于低的网络隔离，不仅网络流量影响网络性能，而且处理器和内存利用率也影响网络性能。因此，在虚拟化场景中，所有这些被监测的度量元都是重要的［FER 10b］。此外，网络管理操作（如虚拟机迁移）也影响网络设备操作［PIS 10］。

De Lucena 和 Moura［DE 09］分析网络流量，焦点是基于报文流的观测数据。作者以一个 4 元组（源 IP 地址、目的 IP 地址、源传输端口和目的传输端口）来定义流。流方法定义各种类型的异常，如 DoS、配置失效和瞬态群集。在 ADAGA 系统中，各流不是以常规方式归组的。该系统依据服务将包放在一起，因为目的是从网络功能的角度来管理网络，考虑到多元论网络方法认为在未来互联网中每项服务一个虚拟网络［MOR 09］。因此，在 ADAGA 系统中，包组考虑到协议号和目的传输端口，因为它们是定义流服务的报文特征。

与异常检测有关的几项工作给出观测间隔量级单位为分钟或数十分钟［BRU 00，DE 09，SIL 10a，SIL 10b］，这降低了处理和存储需求。但是，这个幅度的长时间间隔不支持对重要异常的一次快速响应，这种异常是在一个短时间段中发生的。ADAGA 系统提供观测间隔大约为 15s，因此支持对异常的快速检测和响应。采用原型实施的这些试验，给出令人满意的结果，因为收集和分析的处理是实时完成的，且 40 个不同特征的分析采用一个 Intel Core i7 950 处理器需时 10^{-4}s。除了采样间隔外，入侵检测系统的另一项重要特征是报文采样率。如果它们要评估所有报文，则每报文处理系统要求高的处理和存储负载。出于这个原因，几项工作实施报文采样［SIL 10b，SOU 05，BAR 02］。不过，报文采样在观测结果上引入扭曲、噪声和平滑［BRA 06］。最近的提议解决这个问题的方法是，通过对可能是异常的报文做标记以在其他网络设备中进一步分析［ALI 10］，或通过比随机采样更高效的过滤器［BRA 10］。建议收集流量统计而不是报文采样。因此，以较小的处理额外负担考虑所有报文。

遵循 Dobson 等［DOB 06］描述的自治管理过程链，即收集、分析、决策和动作，异常检测要求发现异常的根源。从网络的最新观测中得到根源［SIL 10a，PAR 10］。ADAGA 没有解决发现异常的根源问题，而是发送最新观测数据与所有收集的数据，由其他根源发现机制进行处理。

7.2.2 ADAGA 系统

ADAGA 系统提供在一个虚拟网络环境中收集和分析数据的机制，并支持虚拟化网络的自治管理。所提出系统的目标是检测虚拟网络上的异常，同时激活异常修复机制。图 7.10 给出 ADAGA 的架构，以及各模块的互联关系和被监测系统与管理器间的通信。

本节定义在任何给定时刻表示和组织网络知识的一个本体。网络管理员选择要

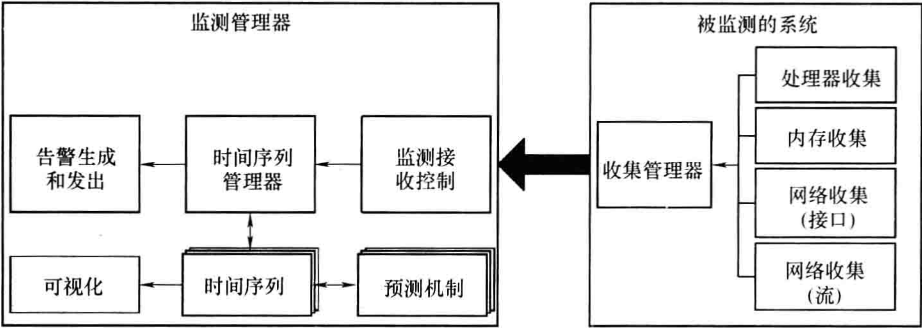


图 7.10 ADAGA 系统架构

被监测的有关特征，且每个这样的特征都被组织为一个时间序列，见下面第 2 部分所定义的。时间序列有一个相关联的预测机制，如下面第 3 部分所述，它测量在这个特征当前测量与过去的偏差，产生下面第 4 部分所述的告警。虚拟化模块产生预测器误差的时间序列图形变化情况，如 7.2.3 节所述。

1. 数据收集和表示

ADAGA 系统通过扩展标记语言（XML）格式的远程数据请求，从网络设备处收集数据。监测管理器查询被监测的系统，它们可以是网络中的物理单元或虚拟单元，如图 7.10 所示。在接收到请求时，每个被监测的系统执行监测代理，触发几个专用代理，得到诸如处理器利用率、内存利用率和网络状态等特定数据。

在一个被监测网络场景［ZIV 05］中，观测度量元是基础性的。原型通过可用的 Linux 操作系统工具（是 ADAGA 中使用的基础系统）和 Xen［BAR 03］（是被选中的虚拟化平台）工具收集观测数据。图 7.11 给出 ADAGA 中网元的一种简化表示结构。

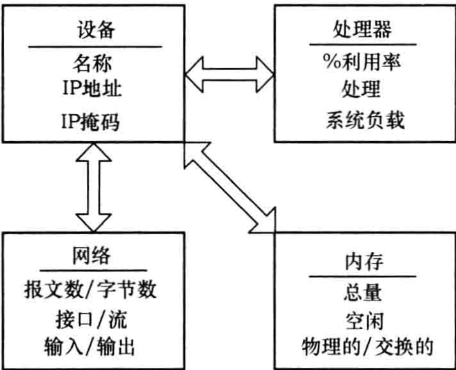


图 7.11 ADAGA 系统中网络组件的表示结构

各网元的建模考虑多核处理器、RAM 机制，即物理内存和虚拟内存，以及几个网络接口。网络设备模型包括设备的识别与位置数据以及与不同处理器的连接，

内存和网络接口模型。每个特征作为独特的时间序列参与到异常检测。在 ADAGA 系统中，为每个特征应用在本节描述的计算。我们宣称，一个有效的异常检测系统必须能够通过分析数据统计信息而不是真实收集的数据，检测一次异常。除了避免与数据报文的隐私有关的问题外，这个特征支持所建议系统具有更有效的处理和存储。

2. 时间序列

依据 Brockwell 和 Davis [BRO 02]，时间序列是一个观测 s_t 的集合，其中每个观测都是在一个特定时刻 $t \in T$ 实施的，其中 T 是测量时间的有限群。一个时间序列和一个常规值集合的区别是，观测间的顺序是重要的。考虑到观测过程，有两种类型的时间序列。在离散时间序列中，观测具有确定的时间，且是在特定时刻实施的。在连续时间序列中，观测是在一个时间间隔过程中连续实施的。在本项工作中，使用离散时间序列。

已观测时间顺序定义集合 T。可考虑观测之间的采样间隔遵循中心在 15s 的泊松分布。依据 Paxson [PAX 98]，一个固定的采样间隔可导致观测中的扭曲，因为它可能与一个不可预测的事件同步，且它不能正确地观测网络上的周期性行为。这些问题降低检测准确度或隐藏异常。

在时间序列中，在任何给定的特定时刻 t，基于序列的过去（s_1，s_2，…，s_t），可预测序列的下一个值 s_{t+1}。依据这个预测和在时刻 $t+1$ 处观测的真实值，则可将预测器误差定义为

$$\varepsilon_t = |\hat{s}_{t+1} - s_{t+1}| \tag{7.2}$$

如果这些误差大于在预测中确定的容忍度，则 ADAGA 触发告警生成模块，见下面第 4 部分所述。

时间序列管理模块在线地以接收到的观测数据馈入和控制时间序列。这个模块定义每个路由器的被监测特征、每个特征的预测器参数和特征时间序列上新值的插入。

3. 预测机制

预测机制确定每个时间序列的下一个值。考虑到两种预测机制：简单的和被很好使用的预测器，称作指数平滑机制，以及 Holt-Winters 季节性的（Holt-Winters seasonal）机制，它考虑时间序列的趋势和季节性成分。指数平滑预测机制是计算一个时间序列中下一个值的一种简单的算法，它基于序列历史的移动平均 [BRU 00]。为计算每一个接下来的值 $\hat{s}_{t+1}$，表示为当前测量值 s_t 和为当前值计算的预测值 $\hat{s}_t$ 的表达式为

$$\hat{s}_{t+1} = \alpha s_t + (1-\alpha)\hat{s}_t \tag{7.3}$$

式中，$\alpha \in [0, 1]$ 和 $\hat{s}_1 = s_1$。参数 α 是在序列历史关系中当前值的权重。因此，α 的值越大，序列过去对预测值计算的影响就越小。依据 De Lucena 和 Moura [DE 09]

的分析，在计算机网络场景上，参数 α 的合适值必须小于0.1。因为从 $t \geqslant 2$ 开始实施的递归，序列过去的影响随着时间而降低，遵循下式：

$$\hat{s}_t = \left[\sum_{j=0}^{t-2} \alpha(1-\alpha)^j s_{t-j} \right] + (1-\alpha)^{t-1} s_1 \tag{7.4}$$

因此，图7.12表明，除了第一个值之外，每个过去观测值对预测计算的影响指数性地降低。因此，初始条件强烈地影响结果，如图7.12a所示。因此，为降低初始条件对预测值的影响，这项工作对序列值应用一次变换，使

$$\forall s_t, s_t = s_t - s_1 \tag{7.5}$$

那么，初始配置将是 $\hat{s}_1 = 0$，没有预测误差，如图7.12b所示。

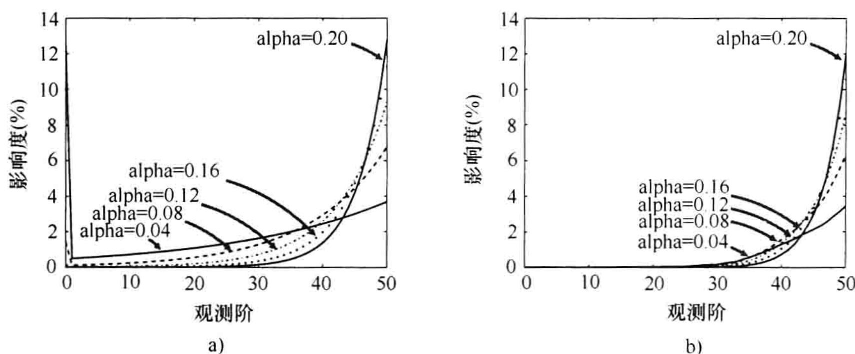


图7.12 在针对传输的报文时间序列的 $t=50$ 观测的预测中，过去观测的影响
a）收集的数据 b）应用所提出的变换收集数据

指数平滑机制不适合呈现周期性行为（称作季节性）的时间序列，因为它假定线性序列值，并以序列历史的移动平均逼近下一个值。Holt-Winters 季节性机制是良好地适合于季节性时间序列的一个预测器。Brutlag [BRU 00] 为计算机网络的行为定义了一个季节性模型，在早晨时段比在夜晚时段有较多的活动。

Holt-Winters 季节性机制将时间序列分解为趋势、季节性和噪声。这些成分中的每个成分可被处理为指数平滑方法的一个变化。有两种方法将这些分量汇聚成一个预测值：加性的和乘性的 [KOL 99]。当周期的统计变化不依赖于序列时，这些分量采用加性方法汇聚。在依赖于序列时，这些分量是乘性的。De Lucena 和 Moura [DE 09] 声称，对于计算机网络，由各分量组成的加性给出较好的结果，因此，下一个值的预测是

$$\hat{s}_{t+1} = R_t + T_t + P_{t+1-m} \tag{7.6}$$

式中，T_t表示时间序列的趋势；P_{t+1-m}是周期性分量；m 是季节性周期；R_t是序列汇聚噪声。

那么，预测方程给定为

$$R_t = \alpha(s_t - P_{t-m}) + (1-\alpha)(R_{t-1} + T_{t-1}) \tag{7.7}$$

$$T_t = \beta(R_t - R_{t-1}) + (1-\beta)T_{t-1} \tag{7.8}$$

$$P_t = \gamma(s_t - R_t) + (1-\gamma)P_{t-m} \tag{7.9}$$

式中，α、β 和 $\gamma \in [0,1]$ 表示被预测值每个分量的平滑常数。

类似于指数平滑机制，这些参数代表被预测值计算上过去序列的权重。常数值越大，则过去分量对预测的影响就越小。

4. 告警生成

当被预测误差大于计算得到的可接受误差时，产生告警。为每个新收集的观测，重新计算可接受的误差，并定义为

$$\varepsilon_t = \delta \Psi_t \tag{7.10}$$

式中，δ 是可接受误差的一个幅度常数；Ψ_t 依据预测器算法加以定义。

Brutlag［BRU 00］声称，δ 的最优值属于区间［2,3］。ADAGA 使用 $\delta=2$，因为提出一个敏感系统，且 $\delta=2$ 产生一个较小的接受误差。

对于指数平滑机制，考虑到观测窗口和下一个值预测，Ψ_t 是值的标准差。对于 Holt-Winters 季节性机制，Ψ_t 是

$$\Psi_t^{\mathrm{HOLT}} = \gamma(|s_t - \hat{s}_t|) + (1-\gamma)(\Psi_{t-m}^{\mathrm{HOLT}}) \tag{7.11}$$

ADAGA 系统提出带有告警累积发送的一种告警控制法。那么，为清除准时的系统告警，并不发送所有产生的告警。在 ADAGA 中这种方法的实现，使用一个滞后来定义所产生告警的发送。如果系统在观测过程中检测一次异常，这个告警并不立刻发送。仅在产生 η 次告警之后，发送一次告警。η 的值是由网络管理员定义的计数器阈值。如果系统在得到 η 的值之前停止检测异常，则不报告异常，且告警累积计数器设置为零。

发出一次告警，意味着将一个报告发送到决策系统，并作用于网络。这个报告由网元的所有特征的一组最近 15 次观测（产生告警）、预测器值和真实观测值的信息以及产生告警的特征组成。

7.2.3　异常系统评估

评估 ADAGA 系统，确定它检测异常的容量。评估考虑到假阳性和假阴性率。假阳性意味着被错误发出的告警（因为没有异常），而假阴性由不产生告警的异常时刻观测所刻画，因此包括在告警产生滞后时段过程中累积的告警。

开发了一个原型，以图 7.13 所示的测试床场景来分析 ADAGA 系统。监测管理器执行时间序列管理、预测和告警产生。被监测系统是用于一个真实网络中的一台无线网络路由器。所收集的信息是周期性的，一个收集间隔后跟中心在 15s 的泊松分布［PAX 98］。

在大约 2 天的监测之后，两个节点开始使用安全外壳（SSH）协议两次连续地将一个 15GB 文件上传到路由器，使路由器过载并避免远程访问。目标是导致必须由 ADAGA 检测的一次异常，结果是发送告警。此外，评估每个特征对这次异常检

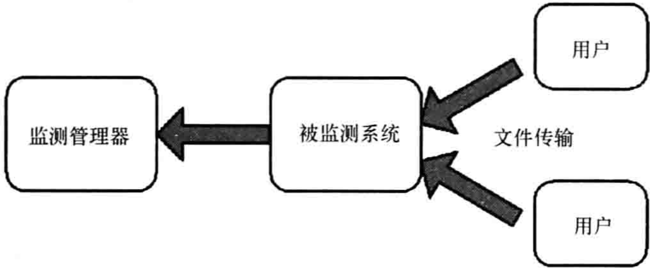


图 7.13 在 ADAGA 中用于假阳性和假阴性分析的测试床场景

测的影响。和预料的一样，网络特征的时间序列，如接收接口中的网络流量和传输控制协议（TCP）端口 22 处的流，来检测异常，而其他特征（如运行进程数和内存利用率）则不能检测异常。观察到，当系统负载变量给出一个特别行为（如大量峰值）时，则检测到一次异常。系统负载受到文件传输的影响，原因是所实施的上传诱发路由器中的网络和磁盘 I/O 负载。

下面介绍结果。

SSH 服务中 TCP 报文接收的观测序列如图 7.14 所示。图 7.14a 给出指数平滑机制的结果，而图 7.14b 给出 Holt-Winters 季节性机制的结果。在图的顶部，有真实的观测（实线）和机制预测的值（虚线）。由这个图观察到，指数平滑机制预测的值容易遵循真实值变化趋势，这使异常检测更加困难，即使对这个幅度的异常的检测也是如此。不同的是，Holt-Winters 季节性机制在跟随被观测值方面存在更大的困难，这给出较好的异常检测效率。当一个预测器能够跟随突然的改变时，它将有检测异常方面的低效率，这被定义为突然改变。图的底部给出与异常检测直接发生联系的值。实线代表机制的预测误差［依据式（7.2）得到］，虚线给出可接受的预测误差［由式（7.10）计算得到］。为得到最佳的可视化效果，各图仅给出异常发生时的时间区间。

在图 7.15 中，观察到两个被监测特征的过去行为。图 7.15a 中在每次数据收集之前 5min 内被监测系统的平均负载，图 7.15b 给出处理器利用率的百分数。由图形分析，观察到特征（如系统负载）容易受到网络流量异常的影响。这个影响确认了多维网络设备监测的重要性。不同特征的异常信号对于跟踪异常的原因是有用的。诸如处理器利用率的特征没有受到所产生异常的影响，如图 7.15b 所示。

对于所有收集的观测，当没有异常时，假阳性率计算发出的告警数。此外，假阴性率是对所有收集的观测，在异常过程中没有发出的告警数。

和预料的一样，相比于指数平滑机制，Holt-Winters 季节性机制给出较好的结果。图 7.16 给出变化 η 参数时，预测机制的几个参数配置的假阳性和假阴性百分比，η 代表在发出报告之前累积的告警量。在 α 等于 0.05、0.10 和 0.15 时，评估指数平滑机制。对 Holt-Winters 季节性机制的 α、β 和 γ 参数，应用同样的值，得

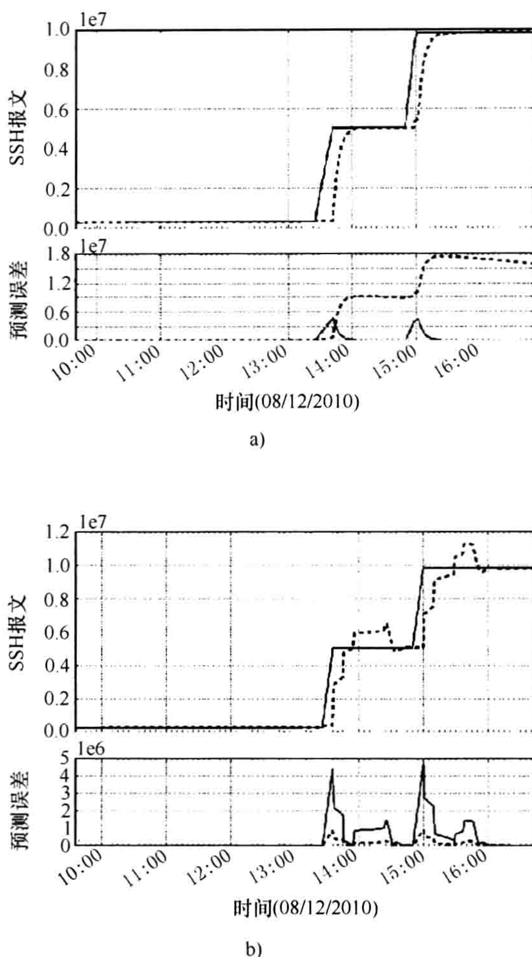


图7.14　对于每种机制，在SSH服务中接收的TCP报文的时间变化趋势（观测窗口为2000个样本，α、β和γ都等于0.1）

a）指数平滑机制　b）Holt-Winters季节性机制

到图7.16c和图7.16d所示的27条曲线。图7.16中所有图形都是对由2000个样本组成的一个观测窗口得到的。针对观测窗口的其他尺寸实施的测试，给出被诱发异常检测中的类似结果。

虽然指数平滑机制给出图7.16a所示低的假阳性率，但它不能检测良好生成的异常。另外，对于所有被评估的参数配置，Holt-Winters季节性机制给出一个不错的效果，即假阳性率低于2.5%，对于所定义的告警累积合适的假阴性率。在异常间隔中，收集40个样本。表7.2给出假阳性率，而表7.3给出Holt-Winters季节性机制所有被评估参数配置的假阴性率。各行是每个α值的假阳性值和假阴性值，各列给出依据β的值分类的每个γ值的假阳性和假阴性值。可得出结论，在不同配置的结果之

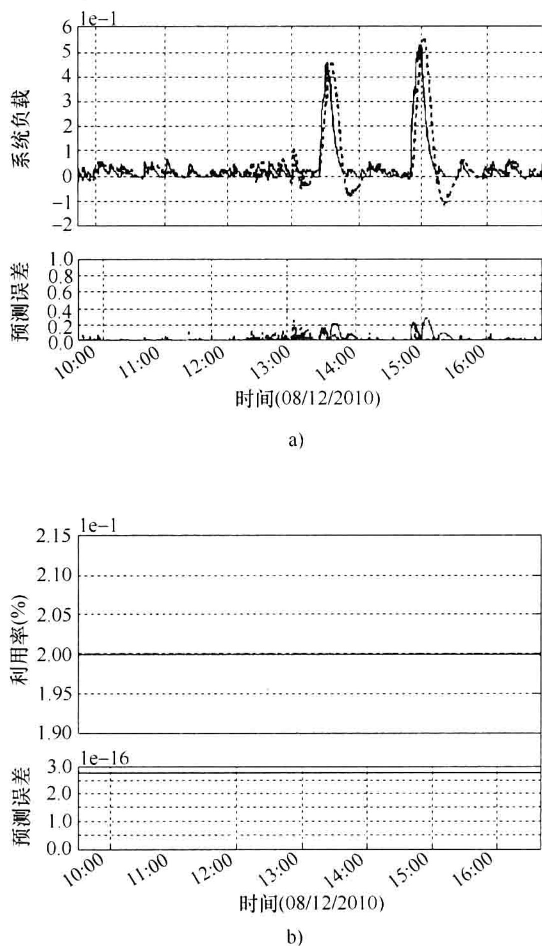


图 7.15 对于 Holt-Winters 季节性机制，不与网络流量直接相关的两个特征的时间变化趋势（观测窗口为 2000 个样本，α、β 和 γ 都等于 0.1）

a）在每次观测之前 5min 内系统的平均负载 b）路由器上处理器利用率的百分数

间存在较大的相似性。不过，对于较小的β值，观察到一个小的增益。参数β与序列趋势有关。因此，在时间序列趋势上预测器接受剧烈变化次数越多，即对较大的β值，则预测器就能够越快地针对异常进行调节，结果，成功率就越小。

表 7.2 对于图 7.16c 所示的 α、β 和 γ 配置以及 η = 0.5 时假阳性率的比较

（各值表示为百分比）

β	0.05			0.10			0.15		
$\alpha \backslash \gamma$	0.05	0.10	0.15	0.05	0.10	0.15	0.05	0.10	0.15
0.05	1.69	1.34	2.19	1.64	1.49	1.81	1.44	1.85	1.59
0.10	1.33	1.19	2.20	2.35	1.85	2.28	1.55	1.93	2.42
0.15	1.80	1.59	1.65	2.28	1.74	1.88	2.01	1.68	2.32

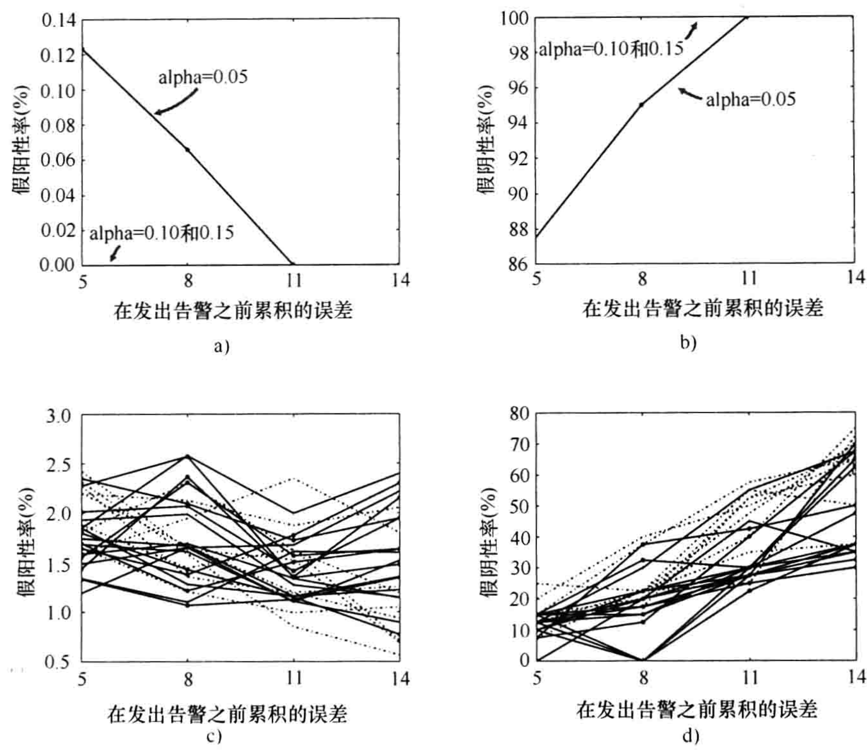


图 7.16　在 SSH 服务中接收报文的每种机制特征的分析，此时改变参数 α、β 和 γ（观测窗口尺寸是 2000 个样本）

a）指数平滑机制的假阳性率　b）指数平滑机制的假阴性率

c）Holt-Winters 季节性机制的假阳性率　d）Holt-Winters 季节性机制的假阴性率

表 7.3　对于图 7.16c 所示的 α、β 和 γ 配置以及 $\eta=0.5$ 时假阴性率的比较

（各值表示为百分比）

β	0.05			0.10			0.15		
$\alpha \setminus \gamma$	0.05	0.10	0.15	0.05	0.10	0.15	0.05	0.10	0.15
0.05	15.0	12.5	7.5	10.0	15.0	12.5	7.5	15.0	10.0
0.10	12.5	15.0	25.0	0	12.5	15.0	15.0	10.0	15.0
0.15	7.5	12.5	15.0	12.5	10.0	15.0	15.0	10.0	20.0

7.3　小结

多元论方法由专用虚拟网络组成，在一个独特的硬件（提供不同 QoS）上，提供不同的专用或通用被隔离的虚拟网络。也提出一个引导平面，为处理网络中出现

的所有问题（如拥塞、故障和 QoS 提供），必须要管理和控制所有的虚拟网络。引导平面使用知识平面提供的信息，自治地执行管理和控制规程。管理和控制整个网络是一项巨大挑战，我们声称，共置视图解决方案有助于这些活动。共置视图是受限于其邻居关系和周边环境的整个网络的一个部分视图。本章给出两个管理和控制系统。

提出的第一个系统基于一个动态资源分配系统的开发，它分析虚拟路由器的资源利用率概要，并基于 QoS 度量元和 SLA 协议提供资源的公平共享。为虚拟化网络环境的 SLA 控制，开发了一个高效的模糊控制器，其中隔离特权表示一项巨大挑战。得到的结果表明，所提出的系统是高效的，为比较适合网络资源控制。网络管理员可容易地插入反映其个人和定性决策战略的规则。得到的结果表明，通过惩罚违反规则的不当行为路由器，系统高效地控制 SLA。此外，惩罚等级取决于系统负载和隔离等级。在试验中，系统成功地以自适应方式限制 SLA。当网络处在有充足空闲资源的一个状态时，系统应用轻微惩罚，在关键时刻，系统应用严厉惩罚。结果也表明，控制器在 5s 内调节资源使用。在一个低负载条件下，系统在 40s 内收敛。此外，监测和管理在控制域内仅产生小的额外负担，这对应于每个被管虚拟路由器单个处理器的 5%。

提出的第二个系统将焦点放在监测和预测技术上，它监测环境，为知识平面提供合适的和最新的信息，也监测路由器不当行为。开发了 ADAGA 网络系统，它从虚拟网络收集和分析数据。ADAGA 的主要优势是多维监测、低的监测间隔（提供低的响应时间）和避免每报文处理战略。通过考虑计数器和统计量而不是数据报文，所提出的系统考虑时间序列并最小化额外负担。结果表明，指数平滑机制不适合计算机网络预测，因为这些网络呈现出季节性变化。假阳性和假阴性结果表明，Holt-Winters 季节性机制是高效的，因为它以低的假阳性率和假阴性率检测网络异常。

本章中给出的这两个系统帮助用户和代理管理和控制虚拟网络，同时感知更新的需求。第一种系统方法监测和管理资源利用率概要和 SLA，维持网络中的 QoS 并估计 SLA 更新的需要，以便得到每个虚拟路由器的需求。通过产生资源利用率概要，则监测违规和保障 SLA 约束条件就是可能的。第二种系统监测几个参数及其使用，并预测未来行为，检测可能的变化，之后触发告警，强制系统在知识平面中保持信息被刷新。

7.4 参考文献

[ALI 10] ALI S., HAQ I., RIZVI S., *et al.*, "On mitigating sampling-induced accuracy loss in traffic anomaly detection systems", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 3, pp. 4–16, 2010.

[BAR 01] BARFORD P., PLONKA D., "Characteristics of network traffic flow anomalies", *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, ACM, pp. 69–73, 2001.

[BAR 02] BARFORD P., KLINE J., PLONKA D., *et al.*, "A signal analysis of network traffic anomalies", *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, ACM, pp. 71–82, 2002.

[BAR 03] BARHAM P., DRAGOVIC B., FRASER K., *et al.*, "Xen and the art of virtualization", *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, ACM, pp. 164–177, 2003.

[BRA 06] BRAUCKHOFF D., TELLENBACH B., WAGNER A., *et al.*, "Impact of packet sampling on anomaly detection metrics", *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ACM, pp. 159–164, 2006.

[BRA 10] BRAUCKHOFF D., SALAMATIAN K., MAY M., "A signal processing view on packet sampling and anomaly detection", *INFOCOM, 2010 Proceedings IEEE*, IEEE, pp. 1–9, 2010.

[BRO 02] BROCKWELL P., DAVIS R., *Introduction to Time Series and Forecasting*, Springer Verlag, 2002.

[BRU 00] BRUTLAG J., "Aberrant behavior detection in time series for network monitoring", *Proceedings of the 14th USENIX Conference on System Administration*, New Orleans, LA, USA, 3–8 December 2000.

[DE 09] DE LUCENA S., DE MOURA A., "Análise dos Estimadores EWMA e Holt-Winters para Detecção de Anomalias em Tráfego IP a partir de Medidas de Entropia", *CSBC2009*, 2009.

[DOB 06] DOBSON S., DENAZIS S., FERNÁNDEZ A., *et al.*, "A survey of autonomic communications", *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[FER 10a] FERNANDES N.C., DUARTE O.C.M.B., "XNetMon: Uma Arquitetura com Segurança para redes virtuais", *Anais do X Simpósio Brasileiro em Segurança da Informação de Sistemas Computacionais - SBSeg10* Fortaleza, CE, Brazil, pp. 339–352, October 2010.

[FER 10b] FERNANDES N., MOREIRA M., MORAES I., *et al.*, "Virtual networks: isolation, performance, and trends", *Annals of Telecommunications*, vol. 66, no. 5–6, pp. 339–355, 2011.

[KEC 01] KECMAN V., *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, MIT Press, 2001.

[KEL 09] KELLER E., LEE R., REXFORD J., "Accountability in hosted virtual networks", *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, ACM, pp. 29–36, 2009.

[KOL 99] KOEHLER A., SNYDER R., ORD J., "Forecasting models and prediction intervals for the multiplicative holt-winters method", *Monash Econometrics and Business Statistics Working Papers*, 1999.

[MEN 06] MENASCÉ D., BENNANI M., "Autonomic virtualized environments", *2006 International Conference on Autonomic and Autonomous Systems*, ICAS 06, , IEEE, p. 28, 2006.

[MEN 10] MENG X., PAPPAS V., ZHANG L., "Improving the scalability of data center networks with traffic-aware virtual machine placement", *INFOCOM, 2010 Proceedings IEEE*, IEEE, pp. 1–9, 2010.

[MOR 09] MOREIRA M., FERNANDES N., COSTA L., *et al.*, "Internet do futuro: Um novo horizonte," *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC* 2009, pp. 1–59, 2009.

[PAR 10] PAREDES-OLIVA I., DIMITROPOULOS X., MOLINA M., *et al.*, "Automating root-cause analysis of network anomalies using frequent itemset mining", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 467–468, 2010.

[PAT 07] PATCHA A., PARK J., "An overview of anomaly detection techniques: existing solutions and latest technological trends", *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[PAX 98] PAXSON V., "On calibrating measurements of packet transit times", *SIGMETRICS/PERFORMANCE: Joint International Conference on Measurement and Modeling of Computer Systems, Madison*, WI, 1998.

[PIS 10] PISA P., FERNANDES N., CARVALHO H., *et al.*, "OpenFlow and Xen-Based virtual network migration", *The World Computer Congress 2010 – Network of the Future Conference*, Brisbane, Australia, pp. 170–181, September 2010.

[SIL 10a] SILVEIRA F., DIOT C., "URCA: pulling out anomalies by their root causes", *INFOCOM, 2010 Proceedings IEEE*, IEEE, pp. 1–9, 2010.

[SIL 10b] SILVEIRA F., DIOT C., TAFT N., *et al.*, "ASTUTE: detecting a different class of traffic anomalies", *Proceedings of the ACM SIGCOMM 2010 Conference*, ACM, New Delhi, India, September 2010.

[SOU 05] SOULE A., SALAMATIAN K., TAFT N., "Combining filtering and statistical methods for anomaly detection", *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, USENIX Association, p. 31, 2005.

[WAN 07] WANG Y., VAN DER MERWE J., REXFORD J., "VROOM: virtual routers on the move", *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, Citeseer, 2007.

[WOO 07] WOOD T., SHENOY P., VENKATARAMANI A., *et al.*, "Black-box and gray-box strategies for virtual machine migration", *Proceedings of Networked Systems Design and Implementation*, Cambridge, MA, USA, April 2007.

[XU 08] XU J., ZHAO M., FORTES J., *et al.*, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches", *Cluster Computing*, vol. 11, no. 3, pp. 213–227, 2008.

[ZAD 00] ZADEH L., "Fuzzy sets*", *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[ZIV 05] ZIVIANI A., DUARTE O., "Metrologia na Internet", *Minicursos do XXIII Simpósio Brasileiro de Redes de Computadores*, SBRC, pp. 285–329, 2005.

第 8 章　系统架构设计

未来互联网应该同时支持多个网络，每个网络都有其自己的协议栈和管理方案，这要确保网络核心的巨大灵活性。据此，本章描述支持创建一个可编程网络核心的一种架构。目标是使用当前的虚拟化技术来设计虚拟化基础设施。对此，使用三种不同技术：Xen hypervisor、OpenFlow 交换机架构和这二者的一个组合体，它集成采用机器和网络虚拟化技术。

Xen 和 OpenFlow 平台的使用利用到这两个机器和网络虚拟化平台的优势。OpenFlow 网络为在没有报文丢失的流迁移和虚拟网络重新映射，提供广泛的支持，而 Xen 提供比较灵活的报文处理（由于使用虚拟机）。思路是使用 OpenFlow 管理流，使用 Xen 提供对网络和路由报文的控制。这样一种方法称作 XenFlow，且它得到一个灵活的和完备的网络管理工具集。

本章给出所开发模块的一个一般视图，支持一致原型的创建，这有利于新网络地址特定问题的开发，如移动性、安全性和服务质量（QoS）提供。在本章中定义了 Xen 和 OpenFlow 平台中开发工具的集成，以及所设计模块如何以一种高效方式相互交互。

此外，本章确定了设计基于策略的架构的引导系统和服务控制需求。具体而言，它为后 IP 网络开发一个网络管理框架，为组成具有自管理能力的一个全局网络，该架构将每个网元关联起来并对其优化。本章也在虚拟化基层内定义一个薄的管理层，该基层是自治的，能够支持自适应监测的共置感知、学习、推理和检测故障，以便提供对系统的自我属性。除此之外，介绍了一个引导系统，并讨论独立自我控制功能的集成。

虚拟网络管理的分布式架构和引导系统原型支持网络基层，进行自我管理虚拟网络。网元的自治管理器具有监测、分析、规划和执行的一个封闭的控制环；针对环路的下一次迭代，这样一个环路将信息馈入知识库。用来测试这个概念，测试床由作为物理网络基础设施之上的路由器的虚拟机组成。也使用基于 Ginkgo 平台的一个多代理系统，开发了所提供架构的一个原型。测试场景将焦点放在虚拟网络的自愈方面，但虚拟网络自我管理的分布式架构是足够通用的，则它可被用作自治计算中的其他功能，如自我配置、自我优化和自我保护。给出一些试验，给出恢复进程的性能。

本章组织如下。8.1 节给出 Xen 和 OpenFlow 平台的总体架构视图。它给出所提供算法如何相互交互的详细视图以及虚拟化管理工具如何使用控制算法。也讨论了辅助功能，如平面隔离和安全通信。8.2 节给出 XenFlow 架构设计，这是组合 Xen 和 OpenFlow 虚拟化平台的一个混合架构；也给出这个新平台主要特征的性能

分析。最后，8.3 节给出本章的结论。

8.1 整体架构设计

本节给出使用 Xen 和 OpenFlow 虚拟化平台所开发原型的架构。给出哪些算法与每个平台相关以及工具如何交互。在本节中也描述了一些模块，是设计用来构造一个一致的工具集的。

8.1.1 Xen 架构

在网络虚拟化方面，为不同网络服务、不同虚拟路由器同时共享一个物理路由器。这个范型的主要特征是报文转发上的隔离和性能。隔离确保独立的虚拟网络操作，在期望虚拟网络操作中防止恶意的或故障的虚拟路由器干扰。Xen [FER 11] 是可被用来在商用计算机中构造虚拟路由器的一个虚拟化平台，每个路由器都有其自己的操作系统和协议栈。但是，Xen 平台没有提供完备的隔离，也给出处理网络输入/输出（I/O）操作上的低性能 [EGI 07]。

为在虚拟网络所使用的每个物理节点提供监测和确保隔离、安全和高性能，给出一个架构设计（见图 8.1 和图 8.2）。图 8.1 给出 Xen 虚拟网络所提供架构中的实体关系，而图 8.2 给出每个物理节点的软件架构。为在不干扰其他服务的情况下确保一个工具的方便升级，对采用一个模块化架构做出决策。此外，向所建议模型添加新特征是容易的，原因是所开发模块的基础是用于基于 Xen 的测试床工具（VNEXT）的虚拟网络管理 [PIS 11]，该工具聚集了所有建议的控制模块。

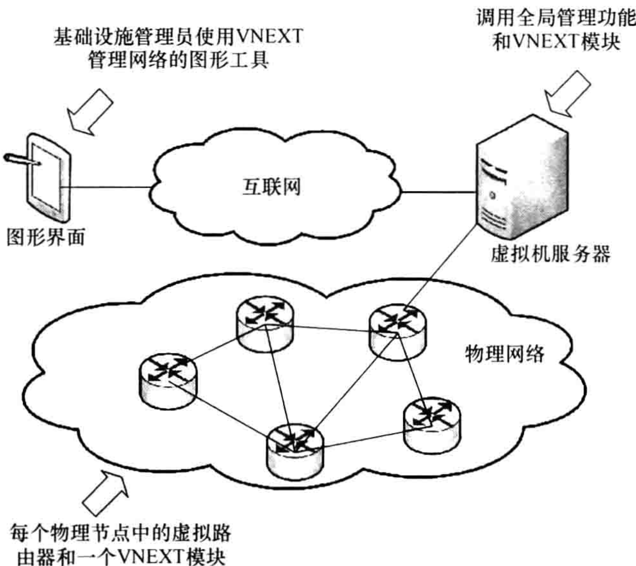


图 8.1　使用 Xen 平台的 Horizon 架构设计

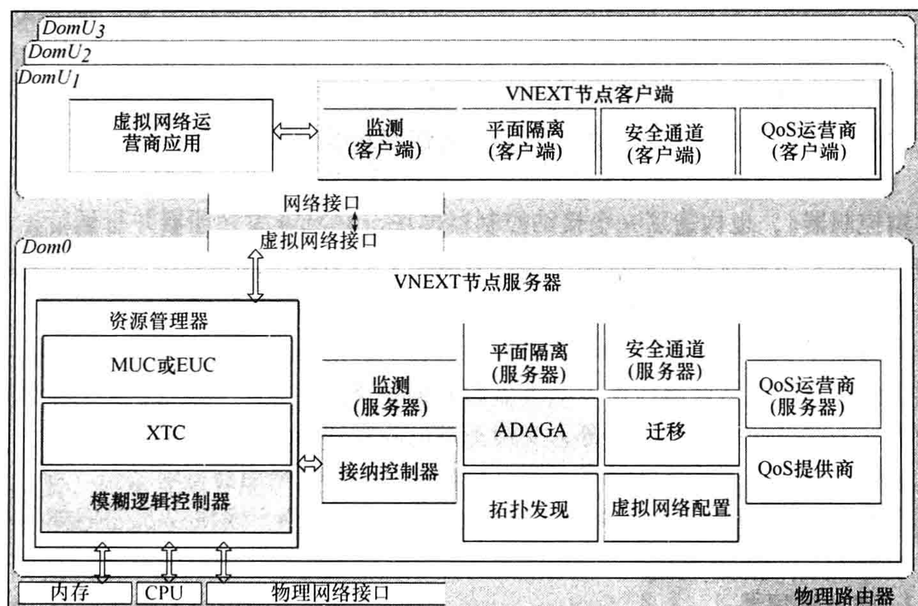


图8.2 使用Xen平台的一个物理节点：域0中的VNEXT服务器和域U中的VNEXT客户端

结构架构是基于一个客户端—服务器模型的，其中虚拟机是位于域0（dom0）中的一台服务器。可用服务是可选择的，并可依据管理员要求加以选择。此外，一些服务仅运行在dom0内，且不干扰虚拟机中的其他服务。

原型中使用的多数客户端—服务器功能是基于平面隔离范型［WAN 08］的，其中数据转发和控制机制是解耦的。因此，诸如路由的控制是在虚拟机［用户域（domU）］中完成的，确保控制机制设计中的灵活性，而报文转发是在称作dom0的一个特权域中实施的，提供准原生性能。平面隔离的主要优势是提供准原生报文转发性能。但是，一个缺陷是这样的事实，即基础设施管理器为所有虚拟机实施一个共同的转发平面。相反，为虚拟网络运营商提供使用或不使用平面隔离范型的选择，平面隔离范型负责在dom0中创建虚拟机转发表的一个有效考虑。QoS运营商模块在dom0内创建虚拟机转发规则（如排队规律和过滤规则）的一个备份。因此，这些模块完成从虚拟机到dom0的数据平面传递。这两个模块使用安全的信道模块，确保domU和dom0之间的安全通信。

监测模块也基于一个客户端—服务器模型，并为dom0和每个虚拟机提供一个监测数据集。除此之外，也描述了迁移［PIS 10］、拓扑发现和虚拟网络配置模块。所有这些模块组成控制网络虚拟化的基本工具集[○]。

○ VNEXT工具发展成为带有安全性的未来互联网测试床（FITS），可在http://www.gta.ufrj.br/fits处得到。

虚拟网络系统自治管理的异常管理（ADAGA）模块检测物理节点和虚拟节点中的异常。ADAGA 与监测模块交互，得到有关 domUs 和 dom0 的数据，并为创建监测虚拟网络的过滤规则而实施动作。

架构的主要模块是资源管理，它动态地控制到虚拟机的资源分配。开发了三个模块［最大使用控制器/高效使用控制器（MUC/EUC）、Xen 吞吐量控制（XTC）和模糊控制器］，也构造了一个接纳控制器模块，它与资源管理器并行运行，评估新的虚拟网络是否能够寄居在同一物理节点中。这个模块为全局虚拟网络分配功能提供支持。

为提供 QoS，向节点架构添加两个不同模块：QoS 运营商和 QoS 提供商。QoS 运营商支持虚拟网络运营商设置 QoS 参数，而不管是否使用平面隔离。QoS 提供商区分由每个虚拟网络提供的服务，支持基础设施管理员为每个虚拟网络设置不同特权。这些模块有利于 QoS 支持的开发，QoS 支持是互联网当前需求之一。

下面提供每个模块的描述，特别是接纳控制器和 QoS 模块以及它们如何相互交互的描述。

1. 资源管理器

原型假定至多使用三个资源管理器，这取决于基础设施管理员希望监测的资源。每个管理器定义一个不同的策略，并控制虚拟化系统中的一个不同参数。

第一个资源管理器模块依据被选中的控制器，选择一个资源策略。有两个选项：MUC［FER 11a］和 EUC。MUC 为每个 domU 预留最小量的资源，并在所有 domUs 间共享剩余的资源。对虚拟网络没有资源使用上界。因此，这项策略允许一个虚拟网络使用比服务水平协议（SLA）中规范的那些资源要多的资源。剩余资源的共享是基础设施管理员设置的一个参数值的一个函数，以便区分虚拟网络。EUC 规定一个最小资源预留率和在一个长时间区间中向每个虚拟网络提供的最大资源量。依据预留参数和网络需求，控制器动态地调节每个虚拟网络的预留参数。因此，EUC 规范每个虚拟网络可使用的最大资源量，提供一个更准确的资源预留方案（由 MUC 提供的）。MUC 和 EUC 与监测模块相关，后者提供实施资源分配控制所需的信息。资源管理器也与虚拟网络配置交互，以便得到和修改虚拟网络资源分配参数。虚拟网络配置模块最初由 VNEXT 虚拟机服务器（VMS）设置，VMS 在网络创建或每个物理节点管理过程中输出由基础设施提供商选择的参数，这确保控制器模块中的一个准确行为。

通过使用一个图形界面，VMS 使基础设施管理员可设置 MUC 或 EUC 的资源参数到每个虚拟节点。默认配置为所有节点假定相同参数，但它也支持每节点配置。

第二个资源管理器模块是 XTC［COU 11］。XTC 是控制一台虚拟机转发报文比特率的一种机制，这将降低它对 dom0 的影响。实际上，为增加或降低虚拟网络的报文转发速率，XTC 动态地调节 Xen 调度器参数，赋予每个虚拟网络或多或少的中央处理单元（CPU）时间。因为 XTC 限制一台虚拟路由器发送的比特率，所以

仅当不使用 VNEXT 架构的平面隔离时，XTC 才是有效的。像所有 VNEXT 模块一样，由 XTC 实施的动作是由 VMS 执行的。为做到这一点，每台物理机器执行称作 XTC 管理器的一个守护进程，它由传输控制协议（TCP）套接字连接到 VMS。因此，通过与 XTC 管理器交换消息，VMS 负责接收用户的请求并采取合适的动作。通过使用 VNEXT 的图形用户界面（GUI），完成由 VMS 实施的 XTC 动作。通过 VMS 发出请求，这个模块提供到 XTC 的一个用户友好的界面。

第三个资源管理器是一个模糊逻辑控制器［CAR 12］。这个控制器将一项互补功能提供给 EUC 和 MUC，使基础设施提供在控制 dom0 中共享 CPU、内存和带宽外的其他参数，如对失效的鲁棒性和机器温度。思路是在虚拟网络环境中提供 SLA 的一种高效控制系统。所提出的系统验证物理资源使用、检索虚拟路由器的实时概要并确保 SLA 需求（得到满足）。控制是基于模糊逻辑的，且它依据系统过载和路由器概要，确定资源分配。控制逻辑惩罚超过已建立 SLA 的虚拟网络。

这个控制器也使用监测模块，并产生概要［表示为直方图（histograms）］，它包含有关的系统信息。

2. 虚拟路由器的接纳控制

虚拟网络接纳控制器仲裁新虚拟路由器到物理机器的访问。寄居于任何物理机器中的虚拟路由器数，影响处理预留的长期容量。实际上，EUC 定义两种资源预留方案：

1）短期速率预留（$R_s[n]$）。这是在短期 I_s内虚拟网络 n 必须要满足的资源预留速率。

2）长期预留体量（$V_l[n]$）。这是在长期时段 I_l过程中应该得到保障的资源总量 $V_l[n]=R_l[n]\cdot I_l$，其中 $R_l[n]$ 是针对虚拟网络 n 的长期平均速率体量预留请求。

实际上，接纳控制机制不是那么简单的，因为不同请求可能请求同样的长期资源。在一项虚拟网络服务请求（要被拒绝的，目的是防止过载物理资源）的概率计算中应该考虑需求概要，该概要包括静态信息（如磁盘和内存尺寸）和动态信息（如吞吐量和 CPU 消耗）。因此，所提出的接纳控制存储表示物理基层资源使用情况的一个直方图集合［FER 12］。在一个监测时段结束处，为每项资源产生一个新的直方图。同一资源的直方图集合对一个时段（如一天）的负载变化进行建模。直方图代表虚拟网络的聚集资源使用情况。具有不同流量概要的网络，得到不同的聚集资源直方图，即使所有虚拟网络的短期和长期预留都相同也是这样的。两个网络可给出相同的 V_l和 R_c，并给出完全不同的行为，这意味着不同的聚集资源需求。

一个监测时段被定义为 $K_{adm}\cdot I_l$秒，其中 K_{adm}是由基础设施管理员选择的一个任意常数。为避免存储和处理过载，所提出的系统随机地选择 K_{rand}，这是在 K_{adm}个长时段之后不被评估的长时段数。此外，接纳控制器并不存储所有的直方图，相反它检查当前直方图和上一直方图之间的差异。为做到这一点，接纳控制器将两个直

方图做归一化处理，并计算这两个直方图之间 y 轴中的最大误差（e_m）。如果条件 $|e_m| > E_{adm}$成立，那么存储当前概要，并开始一个新的直方图。

在一个新的虚拟网络加入物理基层之后，控制器算法估计阻塞概率。这个估计得到的概率被用作一个新虚拟网络可接受决策的一条准则。该算法的输入是基层直方图、每个虚拟网络的长期体量预留（$V_l[\,] = R_l[\,] \cdot I_l$）和每个虚拟网络的平均资源使用情况 $R_{avg}[\,]$。

接纳控制算法是以四步骤完成的：

1）考虑一个直方图，估计聚集资源使用情况；

2）估计需求中的增加如何影响聚集资源使用情况（假定预留得到满足）；

3）估计一个函数，对新虚拟网络资源使用情况进行建模；

4）如果接受新网络，则计算阻塞概率。

步骤 1

步骤 1 的组成是，监测资源使用情况，并像前面描述的那样存储直方图。在这个步骤结束时，该算法知道了每个被监测时段的带宽、CPU 和共享内存的基层直方图。

步骤 2

在这个步骤中，基于所有虚拟网络都在使用所有预留资源的假设，所提出的算法估计基层直方图。思想是确保将总有物理资源满足所有的虚拟网络需求，而不管同时出现的峰值需求是什么情况。基于迁移的解决方案［WOO 09］通常是缓慢的，因为它们依赖于观测一段时间的资源阻塞情况，之后搜索虚拟拓扑和物理拓扑之间的一个新映射，以避免过载物理节点，最后迁移被选中的虚拟节点。这种解决方案可导致报文丢失，造成对基础设施提供商的惩罚。不使用合适的接纳控制的基于迁移的解决方案，可能过载物理节点，导致虚拟网络映射到物理基层中的丢失和震荡。因此，重要的是，在接纳一个新虚拟网络进入一个物理节点之前，估计它对基层的影响。所提出建议的关键思路是，使用如下假设估计资源阻塞情况，即接受新的虚拟网络请求且所有虚拟网络完全地利用相关联的物理资源。

所提出的算法估计需求增加为

$$\Delta = \sum_{n=1}^{N_H} R_l[n] - R_{avg}[n] \tag{8.1}$$

式中，N_H是虚拟网络数。

变量 Δ 估计需求增加对物理基层的影响。为得到这个估计，假定虚拟网络将请求所有预留的资源，且需求没有作为预留请求量级的一个函数而增加。函数 $f_i(t)$对虚拟网络 i 的资源使用情况建模，且这个网络增加的资源使用情况估计由 $f_i(t)+\Delta_i$给定，其中 $\Delta_i = R_l[i] - R_{avg}[i]$。不过，对于所提出算法的每个网络个体地增加其消耗的方式是不相关的，但与每个虚拟网络使用其全部预留时的聚集使用增加的方式是相关的。在得到 Δ［方程（8.1）］之后，控制器更新直方图。首先，依据 Δ，平移直方图。$I[i]$ 是有上界 $L_s[i]$ 的一个直方图时段，$I[i']$ 是包含值

$L_s[i]+\Delta$ 的一个时段，那么

$$L_s[i'-1] < L_s[i]+\Delta \leqslant L_s[i'] \tag{8.2}$$

此外，$H_{sub}(I[i])$ 是在时段 $I[i]$ 中所发生事件数，N_{int} 是直方图中时段数。因此，被平移的直方图 H_{sft} 计算为 $H_{sft}(I[i'])=H_{sub}(I[i])$。$H_{sft}$ 的时段 $I[N_{int}]$ 定义为

$$L_s[N_{int}-1] < I[N_{int}] < \infty \tag{8.3}$$

以便维持固定数量的时段。图 8.3 给出 $N_{int}=10$ 和 $\Delta=1$ 时直方图平移的一个例子。在这个步骤之后，控制器计算被平移子层直方图（PMF_{hist}）的概率质量函数（PMF），假定 x 轴的值由 $L_s[s]$ 给定，如图 8.4 所示。

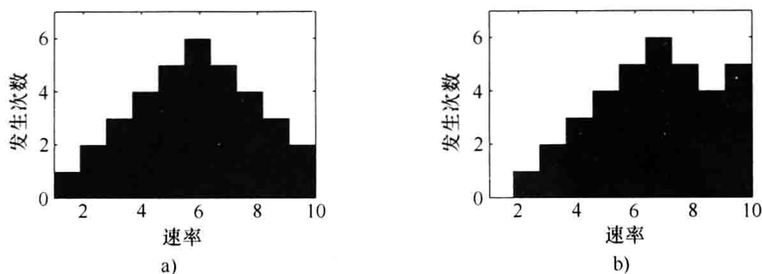


图 8.3　当 $N_{int}=10$ 和 $\Delta=1$ 时基层直方图平移的例子

a）基层直方图（H_{sub}）　b）被平移基层直方图（H_{sft}）

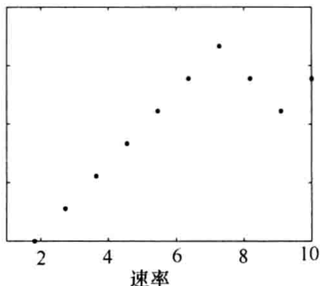


图 8.4　图 8.3b 所示被平移基层直方图的概率质量函数

步骤 3

基于一个预定义分布，控制器估计新虚拟网络的需求。例如，一个乐观的接纳控制器假定一个泊松分布，而一个悲观的接纳控制器假定中心在峰值速率值的一个分布。

依据基层直方图的时段 $I[n]$，表示为新虚拟网络估计的分布（PMF_{new}），假定 x 轴的值由 $L_s[\ ]$ 给定。

步骤 4

控制器算法为每个资源估计阻塞概率。这个概率被用来确定是否应该接受一个新的虚拟网络。依据为新虚拟网络假定的分布（PDF_{new}），所提出算法使用 PDF_{hist}

计算资源阻塞概率。考虑

$$A = \{(x_1, x_2) \mid x_1 + x_2 \geqslant C\}, x_1, x_2 \in [L[0], L[N_{\text{int}}]] \tag{8.4}$$

是一个元组（x_1，x_2）集合，其中 x_1+x_2超过资源容量 C，通过使用式（8.5），控制器计算资源阻塞概率为

$$P_B = \sum_{\forall (x_1, x_2) \in A} PDF_{\text{hist}}(x_1) \cdot PDF_{\text{new}}(x_2) \tag{8.5}$$

因此，所提出算法估计被平移基层直方图中使用资源总和以及新网络超过物理容量的概率。

接受新的虚拟路由器，如果对静态需求存在足够的资源且如果条件

$$\sum_{n=1}^{N} R_s[n] < C \text{ 和 } P_B < P_L \tag{8.6}$$

对所有资源的所有直方图成立，其中 P_L是由基础设施管理员建立的资源阻塞概率界的值。一个小的 P_L值确保低的报文丢失概率值。不过，它也降低资源利用率的效率，且作为一个结果，有利于基础设施提供商。

3. 平面隔离模块

如前所述，为取得高性能，至关重要的是通过 dom0 转发报文。但是，通过将报文转发与路由控制隔离，虚拟路由器不能更新它们的转发表和它们的报文过滤器，原因是它们不能访问 dom0 内存区。系统检测 domUs 中表和过滤器的内容，并在数据平面管理器模块的 dom0 中制作这些内容的一个副本。因此，domU 中的客户端监测转发表和报文过滤器中的变化，且 dom0 中的服务器将每个 domU 中的转发表和报文过滤器映射到 dom0。

domU 中转发表或报文过滤器中的每个变化必须立刻在 dom0 中其副本处得到更新。出于这个原因，在每当有一次控制消息到达时，数据平面管理器客户端就检查 domU 中转发表和报文过滤器中的变化。如果检测到任何差异，客户端通过安全的通信模块将变化传输到 dom0。当数据平面管理器服务器接收到通知一次转发表变化的一条消息时，它搜索那个 domU 的设置以找出要将变化插入到哪个 dom0 表。在报文过滤器更新中，服务器修改接收到的规则，方法是插入规则参数，这些参数规范虚拟网络的特征。这个规程避免一个虚拟路由器在报文过滤器中创建这样的规则：影响其他虚拟路由器中的流量。

4. 安全的通信模块

安全的通信模块在 dom0 和 domUs 之间创建安全的通信信道，在数据传递中提供相互的认证和隐私。要求一条安全的通信信道，原因是数据平面聚集了所有虚拟网络的所有转发表。为保障隔离，隐私和认证是必需的。因为这个模块经常被用来更新 dom0 中的数据平面，所以它必须是轻量的。要求相互认证，是为了确保没有哪个域可伪造一个共同域的身份或 dom0 的身份，以在对应于被攻击域的数据平面中产生误导的信息。

安全的通信模块由两个协议组成：用于交换会话密钥（K_s）的基于非对称密码学的一个协议（见图 8. 5a 中所述）和用于 domU 和 dom0 之间数据安全传输的基于对称密码学的另一个协议（见图 8. 5b 中所述），其中 kp 是私有密钥，Kp 是公开密钥，E（[M]，key）是采用对称或非对称密码学的密钥对消息 M 的加密，Sign（[M]，k）表示消息 M 及其采用 k 的签名，id 是源节点身份。这些协议避免重放攻击，其中敌对域重复老的控制消息来破坏被攻击域数据平面中的信息，使用序列号和随机数，后者是随机选择的数，仅应该在控制消息中使用一次。为增加对抗重放攻击的鲁棒性，每次一个序列号达到一个最大值时，所提供机制也改变会话密钥。这可确保 dom0 和 domU 之间的通信是安全的，因为系统检查真实性、隐私性和数据不可再生性（non-reproducibility）。

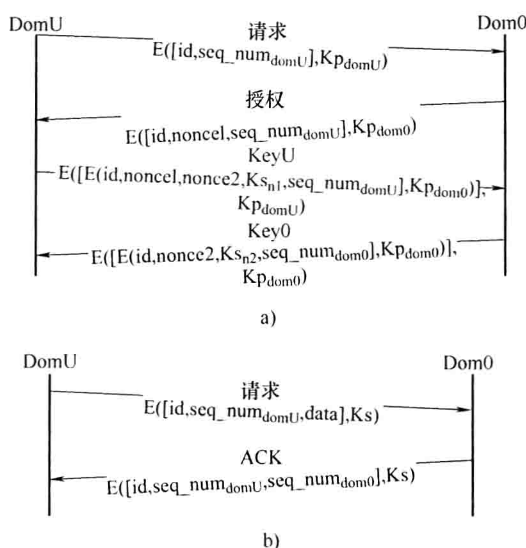


图 8. 5　在 dom0 和 domU 之间创建一条安全信道的消息序列

a）会话密钥建立，$K_s = f(K_{s1}, K_{s2})$　b）更新数据平面的消息交换

5. QoS 提供

所提供架构的一项重要优势是支持 QoS 提供。虽然没有使用平面隔离的一台虚拟路由器可使用一种流量控制机制来实施控制，但它不能确保 QoS。在将流量从物理设备驱动转发到虚拟设备驱动时，一台虚拟路由器对其自己的流量没有控制能力。Xen 架构中的 I/O 操作是在 dom0 中完成的，这是驱动域，结果是，domUs 没有转发操作的完全控制能力。当采用平面隔离时，数据平面被放置在 dom0 中，这支持转发功能和 QoS 支持的细粒度控制。所提供架构为在虚拟网络中和虚拟网络间添加 QoS 规则提供原语，如图 8. 6a 和图 8. 6b 所示，分别是通过虚拟机和通过 dom0 的报文转发。图 8. 6a 详细描述所提供架构的 QoS 模块，其中假定报文转发要通过虚拟机进行。依据这种方案，基础设施提供商可配置一些虚拟网络对物理硬件

的优先级访问。此外，任何虚拟网络运营商可在虚拟机内部区分正被处理的它自己的报文。

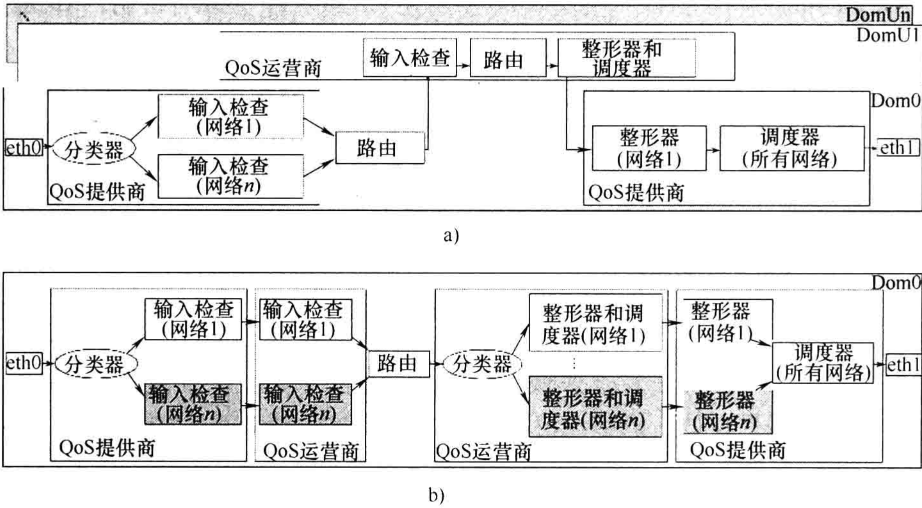


图 8.6　在所提供架构中每个虚拟网络内（QoS 运营商）和虚拟网络间（QoS 提供商）的 QoS 提供

a）假定通过虚拟机进行报文转发的 QoS 提供

b）假定平面隔离范型的 QoS 提供，其中报文转发是排他地在 dom0 中实施的

平面隔离的使用意味着每个虚拟网络的数据平面（以前在 domU 中）现在被放到 dom0 中。因此，QoS 提供必须被修改以便与平面隔离一起工作。结果，必须在 dom0 中准备虚拟网络运营商和基础设施提供商的 QoS 提供方法，如图 8.6b 所示。所提出的架构提供了一个接口，使虚拟路由器可通过 QoS 运营商客户端和 QoS 运营商服务器组件来配置它们自己的 QoS 规则。这些组件保障一个虚拟网络可配置它的 QoS 规则，而不干扰其他虚拟网络的 QoS 规则。

6. 分析

本节分析当与 EUC 控制器一起使用时的访问控制和 QoS 模块。引入这些模块是为了保障对 QoS 需求的支持。

（1）接纳控制模块评估

所提供架构的虚拟网络接纳控制是以 C ++ 实现的。使用仿真在不同流量模式下评估了所提出的算法。将所提供建议与其他方法进行了比较，即 Sandpiper [WOO 09] 和基于蚁群元启发的虚拟网络内嵌算法（VNE-AC）[FAJ 11]。

Sandpiper 是监测数据中心中虚拟机负载的一个系统。当一台物理机器过载（当对资源的请求被阻塞时确定的）时，Sandpiper 寻找一个新虚拟机，该虚拟机能够寄居过载节点的一个或多个虚拟机。Sandpiper 中的接纳控制基于当前虚拟机峰值负载和新物理机器中空闲资源的平均量。如果资源需求的峰值为其累积分布函数

的 95%，或如果它小于或等于空闲资源的平均量，那么接纳新的虚拟网络。

VNE- AC 是一种虚拟网络映射机制，它是 Horizon 项目的组成部分，用于全局虚拟网络的控制。这个算法基于蚁群元启发式方法，确定物理基层上的适当映射。VNE- AC 中建议的接纳控制机制假定资源是静态地属于每个虚拟网络的。结果，这个接纳控制法限制 SLA 规格。

实现了 Sandpiper 和 VNE- AC 接纳控制，将它们与所提供机制进行了比较。依据作者的提议，Sandpiper 峰值速率被选为 $p_k = 95\%$。VNE- AC 预留阈值被设置为 R_l，这是 EUC 中长期体量预留的平均速率（V_l）。为验证这个规程的效率，也评估了 Δ［式（8.1）］对结果的影响，表示为“Hist”。

在评估中，测量了一个物理路由器中可寄居的由每种机制接纳的虚拟网络数。出于简单性考虑，假定所有虚拟网络给出相同的资源预留参数，且对于我们的建议，直方图中时段数为 $N_{\text{int}} = 30$。

也测量了每个试验中被分析物理节点中的理想虚拟网络数。这个理想数被选作保障由基础设施管理员确定的阻塞概率阈值（P_L）不被违规的最大虚拟网络数。因为 Sandpiper 的 $p_k = 95\%$，对于一个公平组合，选择 $P_L = 0.05$。一方面，如果一个接纳控制机制接纳的网络数小于理想数，则浪费了物理机器资源。另一方面，如果接纳控制机制接纳的虚拟网络数大于理想数，则物理节点被过载，导致对基础设施提供商的惩罚。对于接纳高于理想值的多个虚拟网络而言，首选接纳较少的虚拟网络。

为每个试验执行了总共 30 个轮次，分析了以报文数每秒表示的输出链路吞吐量；将给出所关注度量元的标准差。

1）流量模式影响。第一个试验采用以泊松过程建模的流量，评估虚拟网络。每个虚拟网络需求≈100Mbit/s，它也是 R_l 的值。图 8.7a 表明所有机制都接纳理想的虚拟网络数。因此，如果虚拟网络流量呈现一个小的偏差，那么所有被分析机制都能够正确地实施接纳控制。图 8.7b 表明，这个场景中的接纳控制是不连续的：10 个网络不导致阻塞，而 11 个网络导致 100% 的阻塞。

为评估各建议在具有较大流量变化的环境中的情况，也采用由一个 on- off 模型描述的流量模型仿真虚拟网络。在图 8.8 中给出结果。在这个试验中，所有虚拟网络呈现相同流量模式。基于 $\mu = 1/3$ 的一个指数分布产生 on- off 流量。时间间隔是指数分布的。在 on 状态中的流量由 $\lambda \approx 200$Mbit/s 的一个泊松分布进行建模。图 8.8a表明 Sandpiper 和 VNE- AC 比其他建议接纳更多的虚拟网络。不过，这两种机制接纳的虚拟网络大于虚拟网络数（图 8.8b）。Sandpiper 低估了资源占有率，并支持约 10 个网络的接纳。VNE- AC 的行为类似于 Sandpiper。VNE- AC 在图 8.7 和图 8.8 中的场景之间不能做出区分。具有不同数据模式的虚拟网络是同等对待的，产生接纳控制的误导信息。建议接纳接近理想数的多个网络，产生低的阻塞概率和高效的资源使用。

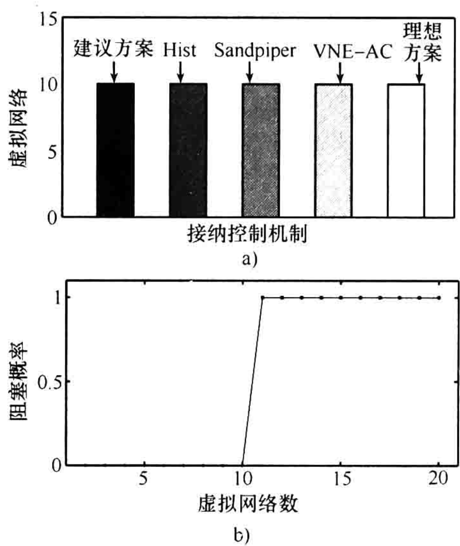


图 8.7　假定虚拟网络具有以一个泊松过程建模的流量和 5% 最大阻塞概率下的接纳控制

a）由每种机制接纳的虚拟网络数　b）依据并行虚拟网络数的阻塞概率

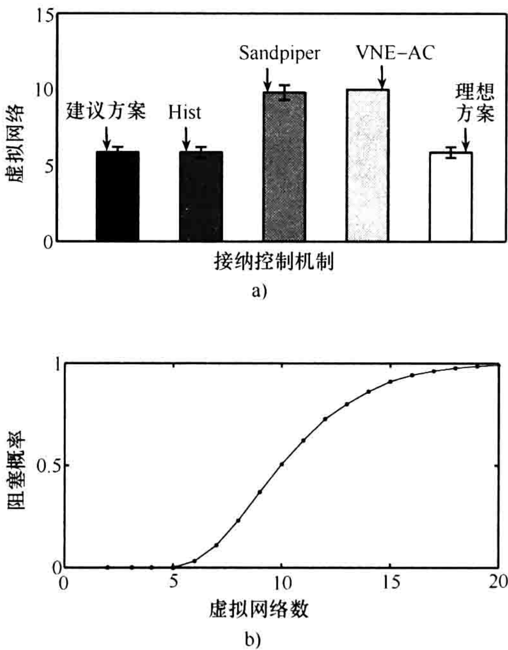


图 8.8　假定具有 on-off 流量的虚拟网络和 5% 最大阻塞概率的接纳控制

a）由每种机制接纳的虚拟网络数　b）依据并行虚拟网络数的阻塞概率

2）Δ 影响。这个试验评估具有变化需求的虚拟网络。在这个场景中，虚拟网络的吞吐量增加到长期体量预留的阈值。当其他虚拟网络做 1/2 预留时，发送新虚拟网络的接纳请求。图 8. 9a 表明算法 Hist 和 Sandpiper 接受所有的网络，因为它们不考虑需求增加。建议和 VNE- AC 考虑变化的需求，并支持接近于理想数的被接受虚拟网络数。

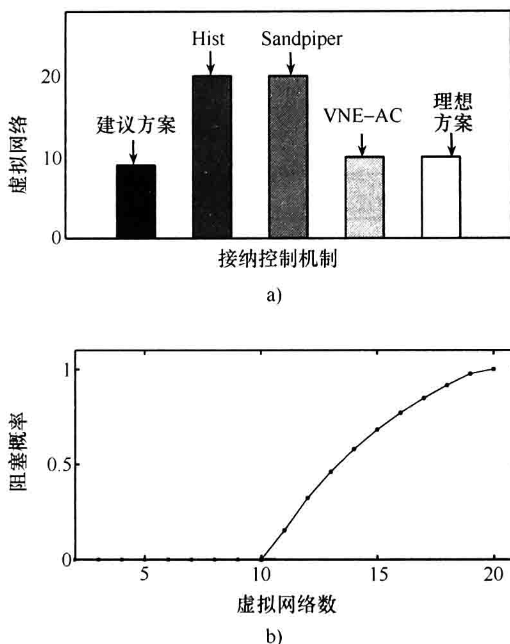


图 8. 9　假定具有增长流量和 5% 最大阻塞概率的虚拟网络的接纳控制

a）由每种机制接纳的虚拟网络数　b）依据并行虚拟网络数的阻塞概率

基于这些结果，可看出所提供接纳控制，是在所有被分析场景中高效的唯一被分析机制，保障接纳大量虚拟网络，而不违反基础设施管理员施加的阻塞概率阈值。

（2）服务质量提供的评估

评估了所提供架构的 QoS 组件 [FER 12]。图 8. 10 给出基础设施提供商使用 QoS 前提的效应。在由两个虚拟网络组成的这个场景中，假定在网络 1 中，其流量面临一个小的转发时延。为输出链路假定如下参数值：对网络 1，$R_s[1]$ = 50Mbit/s 和 $R_l[1]$ = 400Mbit/s；对网络 2，$R_s[2]$ = 100Mbit/s 和 $R_l[2]$ = 600Mbit/s。CPU 和内存资源是均等地在网络之间分割的。网络输入需求分别是 $D[1]$ = 50Mbit/s 和 $D[2]$ = 1Gbit/s。为 $D[2]$ 选择一个大的值，原因是一个大体量的流量可阻止向网络 1 提供 QoS。使用平面隔离的网络 2 的流量，在两台外部机器之间流动。通过虚拟机路由的网络 1 流量，是由一台不同的外部机器产生的，它与网络 2 流量共享输

出链路。

图 8.10 给出带有和不带有所提供 QoS 支持的网络中的往返时间（RTT）值。在第一个场景中，标记有“w/o 优先级”，两个网络有相同优先级，而在第二个场景中，标记有“优先级”，网络 1 流量是优先的。当为一个虚拟网络赋予优先级时，即使不使用资源共享管理器 EUC，对于带优先级的流量，RTT 值也减少 10 倍以上。使用 QoS 支持，确保网络 2 既不超过网络的使用量也不超过 CPU 的使用量，降低了被处理数据体量，由此降低了传输时延。因此，当与没有采用建议和没有优先级的场景比较时，QoS 模块将 RTT 值降低 18 倍以上；当与带有优先级流量但没有资源共享管理器 EUC 的场景比较时，QoS 模块将 RTT 值降低 1.8 倍以上。

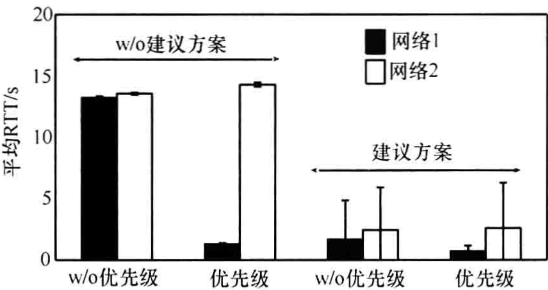


图8.10 假定网络 1 是带优先级的，则依据虚拟网络间的 QoS 参数得到的 RTT

8.1.2 OpenFlow 管理架构

OpenFlow 有不同于 Xen 的一个架构。在 OpenFlow 网络中，控制平面是中心化在一个节点中的，而数据平面则分布在物理 OpenFlow 交换机上。

开发工具 OpenFlow 管理基础设施（OMNI），它被用作集成 OpenFlow 管理模块的基础［MAT 11］。总体架构概图如图 8.11 所示。

由于数据平面的去中心化特性，针对 OpenFlow 网络提出的架构是简单的。在转发节点中不需要修改。被共享的节点没有注意到它们的虚拟化。因此，所提出的功能受限于控制器和 FlowVisor。

OMNI 的原始版本是为与任何 OpenFlow 控制器一起工作而设计的，提供网络视图和虚拟网络间管理功能。OMNI 的扩展版本与 FlowVisor 耦合工作，向基础设施管理员提供管理功能。这个扩展版本为由所有虚拟网络共享的物理资源运行控制算法。

在图 8.11 右侧描述新的 OMNI 模块。在这幅图中，观察到所有控制器与 FlowVisor 交互，FlowVisor 接下来与 OpenFlow 网络交互。OpenFlow 网络是由 OpenFlow 交换机组成的。可选择的做法是，可在这些节点中使用代理，来增加管理性能和添加新的监测功能。各代理也可聚集在中间设备，甚至在基础设施控制器节点

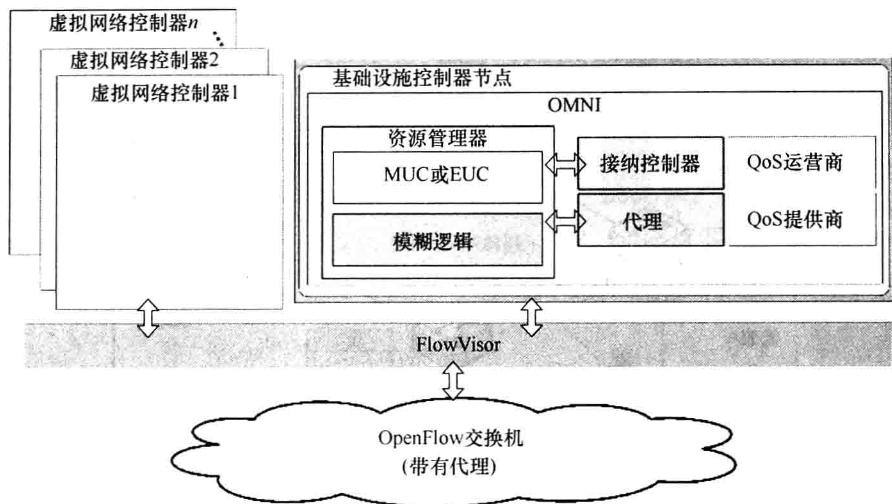


图 8.11　使用 OpenFlow 平台的 Horizon 架构设计

中，以有利于分布式控制功能的性能提升。

FlowVisor 是由 OpenFlow 团队提供的一个接口，在几个控制器间共享 OpenFlow 交换机［SHE 09］。FlowVisor 提供配置物理交换机资源（如内存和队列）的一个接口。但是，它不提供控制算法。采用 Xen 使用的相同控制器算法来控制 OpenFlow 中的资源，原因是这些算法是平台无关的。因此，在 OMNI 中可应用 EUC、MUC、模糊逻辑、接纳控制、QoS 运营商和 QoS 提供商等模块的逻辑，来控制 FlowVisor 资源。也添加代理来模拟一个分布式控制行为。

下面介绍 OpenFlow 管理架构的评估。

评估了 OMNI 的响应时间和操作运行，将由 OMNI 产生的控制报文数与网络操作系统（NOX）产生的控制报文数做了比较，以便估计由 OMNI 产生的控制额外负担。使用运行 OpenvSwitch 软件［WAN 08］（OpenFlow 的一个实现）的个人计算机配置一个试验网络。OpenvSwitch 作为一个内核模块工作，并确保高性能的报文转发。试验场景由 4 台 OpenFlow 交换机、一个 FlowVisor 实体和一个 NOX 控制器组成，如图 8.12 所示。OpenFlow 交换机和 FlowVisor 运行在 Intel Core 2 Duo 计算机上，带有 2GB 内存。控制器运行在带有 4GB 内存的一台 Intel I7 计算机上。在这台计算机上也运行 Ginkgo 代理（控制每台 OpenFlow 交换机的代理）。期望的置信区间是 95% 的置信水平。

第一个试验评估由多代理系统实施的迁移。第一个试验由一条流迁移组成，从由交换机 A、B 和 D 组成的路径迁移到由交换机 A、C 和 D 组成的路径（见图 8.12）。探测流量是一条用户数据报协议（UDP）流，报文尺寸为 1470B，速率从 0.5Mbit/s 变化到 3Mbit/s。在这个场景中，AB 链路的吞吐量受到 200kbit/s 的 OpenFlow 上界约束，而其他链路（BD、AC 和 CD）使用它们的全容量（Mbit/s）。

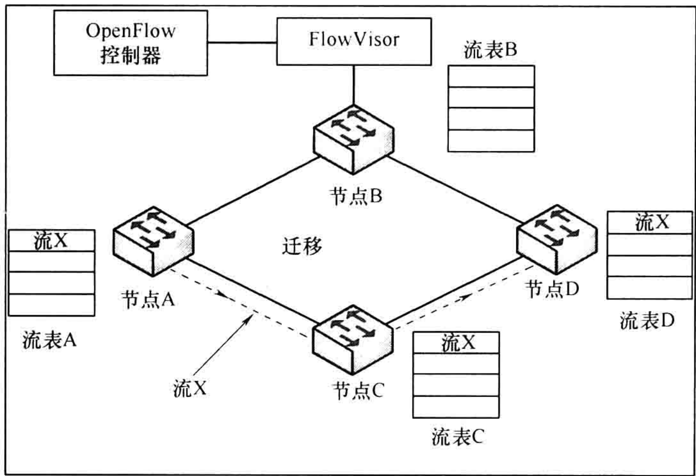


图 8.12 从路径 A-B-D 到路径 A-C-D 的流 X 迁移

因此，当传输速率超过 200kbit/s 时发生报文丢失。测量报文丢失，以便自治地触发流迁移。为就流迁移做出决策，代理验证其被监测交换机的丢失速率，也将局部报文丢失率与其他代理交换得到的丢失率比较。出于比较目的，将报文丢失率和阈值提供给代理作为参数，并依据每个网络进行设置。所开发的代理以 10s 的固定时间间隔感知网络，并仅在三次连续的采样表明报文丢失率大于阈值之后，才迁移一条流。图 8.13a 表明，代理正确地检测流路径内的瓶颈链路，之后迁移该流以避免或降低报文丢失率。因为各代理在固定时间间隔处观测链路中的丢失率，则启动代理和就流迁移做出决策之间的时间是独立于报文传输率的。监测表明，在启动测量规程之后，代理平均 29.4s 触发迁移。

流实例化是在一个 OpenFlow 网络中控制额外负担主要原因之一，因为当一条报文不匹配一台交换机中的任何流时，该报文被转发到控制器，且控制器发送一条命令到交换机。因此，下一个试验评估由 OMNI 引入的控制额外负担，以及 OMNI 应用对流实例化的影响。在使用 NOX 或 NOX + OMNI 实例化新流时，测量穿越一个 OpenFlow 网络的控制报文速率。“NOX” 指代运行其原始应用的一个 NOX 控制器，这些应用收集网络统计量和配置报文转发，而 “NOX + OMNI” 指在 NOX 控制器上运行所有 OMNI 应用。图 8.13b 对于一个变化的流实例化速率比较控制额外负担。对于高达 400 条流/s 的实例化而言，这两个系统之间的差异是可忽略的，原因是 NOX 和 NOX + OMNI 给出几乎相同的控制负载。当实例化速率大于 500 条流/s 时，NOX + OMNI 的控制负载要小于 NOX 的控制负载，但图 8.13c 表明，NOX + OMNI 不能实例化 NOX 那么多的流。采用 OMNI 取得的流实例化速率大于 NOX + OMNI 的速率。同样，因为测试不稳定性，NOX + OMNI 的误差条尺寸（error bar size）增加。实际上，NOX + OMNI 尝试处理比控制器能够处理更多的数据。因为

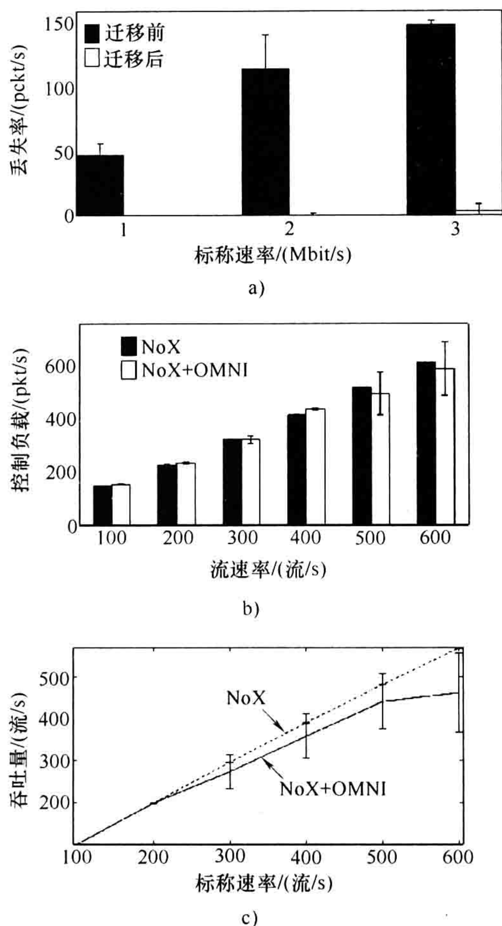


图 8.13 由一个代理调用迁移功能的迁移试验以及 NOX 和 OMNI 之间控制过载比较结果

a）运行代理前/后的报文丢失 b）NOX 和 OMNI 控制负载

c）NOX 和 OMNI 流实例化速率

监测每个资源的 OMNI 间隔是可配置的，增加这个间隔就降低 OMNI 额外负担，这以增加统计测量粒度为代价，导致较高的流实例化速率。

8.2 一个混合的 Xen 和 OpenFlow 系统架构设计

Xen 和 OpenFlow 都有优势和劣势。OpenFlow 给出简单的流配置机制，Xen 在报文处理方面是比较灵活的。这就促动了一种混合架构的定义，强调每个平台的优势，得到一个功能强大的虚拟化平台。采用在前面各节描述的少量修改的管理工具［MAT 11，PIS 11］也应用到这个架构。

在互联网核心中使用新协议和服务，与多数服务提供商的趋势是矛盾的，这是由于这些改变对于正确的网络操作运营代表高风险以及改变平台硬件中涉及的较高成本。在参考文献［FEA 07，RAT 05］中给出在生产网络中支持创新的网络虚拟化建议，这支持被隔离的网络共享相同的物理基层。

网络虚拟化引入一个新的管理原语：虚拟网络的迁移［WAN 08］。在不同语境中使用迁移原语，如物理节点的维护和将逻辑拓扑重新映射到物理拓扑上。物理节点的维护经常要求关闭设备。在路由器的情形中，维护期间导致邻接性丢失，则结果，出现在路由算法收敛过程中的网络失效。为流量管理和能量节省（其中考虑到能量节省，对虚拟节点进行重新组织［BOL 09，ROD 12］），使用虚拟网络重新映射［ALK 12，ALK 11］。迁移也可被用来防止破坏，如在一次拒绝服务（DoS）攻击下的情况。在这个场景中，与处在攻击之下的网络共享相同物理基层的虚拟网络被迁移到其他节点，这就防止了输入链路中的过载。但是，虚拟拓扑的迁移给出巨大挑战，如虚拟链路的重新定位，以及由迁移过程中服务下线时间导致的负面影响的降低。

存在其他建议［WAN 07，PIS 10］，以一种透明方式将逻辑拓扑迁移到网络边缘，而没有报文丢失或中断的连接。但是这些建议是有效的场景，是有限的。在参考文献［WAN 07］和［PIS 10］中，假定链路迁移机制的存在，是独立于节点迁移机制的。它也假定，一台虚拟路由器仅可从一台物理机器迁移到相同局域网（LAN）中的另一台机器。否则，为仿真一个 LAN，应该创建物理机器之间的隧道。

但是，OpenFlow 平台中的流迁移是容易的。Pisa 等给出一种算法，它基于 OpenFlow 网络中一条流路径的定义［PIS 10］。这个建议给出零报文丢失和低的控制额外负担。不过，OpenFlow 迁移不适合路由器虚拟化或流处理系统。这个建议受限于交换式网络。

本节介绍 XenFlow 架构，这是基于 Xen 和 OpenFlow 的一个混合网络虚拟化平台。建议描述一个流处理系统，它支持虚拟网络的迁移，包括节点和链路的迁移。在这个架构中，使用平面隔离范型，之后虚拟路由器被分成两个平面：控制平面和数据平面。控制平面运行在一个 Xen 虚拟机中，负责在给定路由协议决策情况下更新路由表。使用 OpenFlow 实现的数据平面，则负责依据路由策略转发报文。路由策略基于确定的路由，这是由控制平面计算得到的。这项网络虚拟化技术的主要优势如下：

1）带有高度灵活的数据平面的平面隔离；

2）没有报文丢失的迁移；

3）迁移不被限制在局部网络；

4）将逻辑链路映射到一条或多条物理链路；

5）OpenFlow 数据平面使用分布式网络控制；

6）使用相同规程，实施节点和链路迁移。

构造一个 XenFlow 原型，验证系统架构设计。试验结果表明，该系统实施高效的迁移，这是指在迁移过程中没有报文丢失或路由服务中断；系统允许在没有链路丢失或报文转发时延的条件下，实施虚拟路由器和链路的迁移。当将 XenFlow 迁移与 Xen 虚拟机原生迁移比较时，XenFlow 给出零报文丢失，而 Xen 原生迁移丢失大量报文，并在控制平面更新过程中呈现较长的下线时间。

8.2.1 Xen 和 OpenFlow 虚拟化平台的优势和劣势

OpenFlow 是一项交换技术，通过将动作与流关联，支持对报文转发编程。一条流由从帧首部抽取的多达 12 个字段的集合定义，包括来自链路层、网络层和传输层首部［MCK 08］的数据。一台 OpenFlow 交换机的转发表是流表。依据由一个中心式控制器定义的输出动作，流表将一条流与交换机的一个或多个输出端口相关。控制器处理一条流的第一个报文，之后，定义动作。Nox［GUD 08］是一个 OpenFlow 控制器，它作为控制程序和 OpenFlow 网络之间的一个接口。一旦一条报文到达一台 OpenFlow 交换机，该交换机就检查该报文是否匹配任何已经定义的流。如果是，则为那条流定义的动作就应用到该报文。如果不是，则报文首部被发送到控制器，由之从报文中抽取流特征，并在 OpenFlow 交换机的流表中创建一条新流。一个 OpenFlow 网络的一个例子如图 8.14 所示。在 OpenFlow 网络中，迁移是容易实施的，因为仅通过在交换机中重新编程流表就可做到。

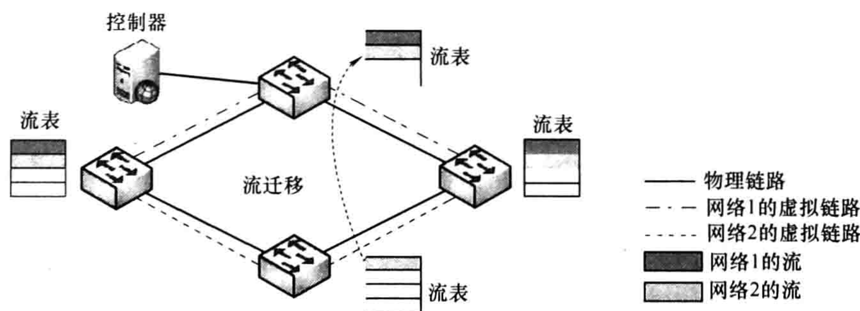


图 8.14　OpenFlow 网络虚拟化：流迁移规程是在另一个交换机集合中重新定义网络流

OpenFlow 网络的两个主要劣势是需要一个中心式控制平面和每跳报文处理的开销。

Xen 是一个个人计算机虚拟化平台，主要用于服务器合并。其架构基于一个虚拟化层，称作虚拟机监测器（VMM）或 hypervisor。Xen 虚拟环境被称作虚拟机或域，并给出它们自己的资源，如 CPU、内存、磁盘和网络访问。也存在一个特权虚拟环境，称作 dom0，它可访问物理设备，并为来自其他域的 I/O 操作提供访问。管理操作也由 hypervisor 实施。图 8.15 给出基于 Xen 的网络虚拟化的一个例子。在

这个场景中，一次虚拟路由器迁移等价于将一台虚拟机从一台物理机器迁移到另一台物理机器。因为路由器实施实时服务，一台虚拟路由器迁移要求最小化报文转发服务下线时间。

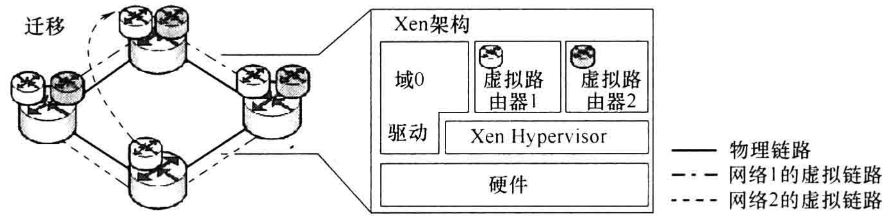


图 8.15 Xen 网络虚拟化：一个网元迁移等价于一次虚拟路由器迁移

Xen 原生迁移［CLA 05］基于虚拟机实况迁移。虚拟机实况迁移组成是，将虚拟机内存从源物理机器复制到目的物理机器。这种迁移被称作实况的，原因是在迁移完成时，在迁移的第一个阶段过程中虚拟机是运行的，它降低了虚拟机的下线时间。因为在源物理机器中虚拟机内存页在迁移规程中可能发生变化，所以这个实况规程使用内存页的一种迭代复制机制。在迭代复制中，被修改的内存页被打上标签，且在下一次迭代中，它们被复制到目的地。这个过程一直重复，直到在上一轮中被修改内存页数小于某个阈值。当虚拟机执行悬挂在源物理机器中时，最近被修改的内存页被复制到目的地，之后，在目的物理机器中恢复虚拟机。针对虚拟路由器迁移的这项建议的一个劣势是，在虚拟机不可用时间过程中（即悬挂和恢复之间消逝时段过程中）的报文丢失。但是，这种机制受限于一个局域网，因为这种方法假定存在一个共享的硬盘，且链路迁移是通过发送 ARP Reply（ARP 应答）报文来实施的。

可使用平面隔离来避免迁移过程中的报文丢失。Pisa 等提出针对 Xen 平台的一种虚拟机迁移方法，它将所有虚拟路由器数据平面复制到 dom0 中［PIS 10］。因此、在不影响路由器和没有下线时间的条件下，实施数据平面迁移。但是，这种解决方案是受限的，因为一次虚拟路由器迁移仅在同一 LAN 中的节点之间进行。因此，迁移范围被限制在离源路由器一跳远的范围。

8.2.2 XenFlow 架构设计

所提出的架构设计将每跳报文处理和分布式控制（由 Xen 平台提供）与快速数据流处理能力（由 OpenFlow 平台提供）的优势组合在一起。一个 XenFlow 网元的架构如图 8.16 所示。每个虚拟机寄居一个不同网络的控制平面。依据由虚拟机指定的转发规则，dom0 中的 OpenFlow 交换机实施报文转发。在这个架构中，一个网元可被定义为一台虚拟交换机（层 2）、一台虚拟路由器（层 3）或一台中间设备（层 3 以上）。由每个网元实施的功能取决于虚拟网络协议栈和应用。

在 XenFlow 系统和 Xen 平台中，物理设备驱动都在 dom0 中，那么虚拟机和物

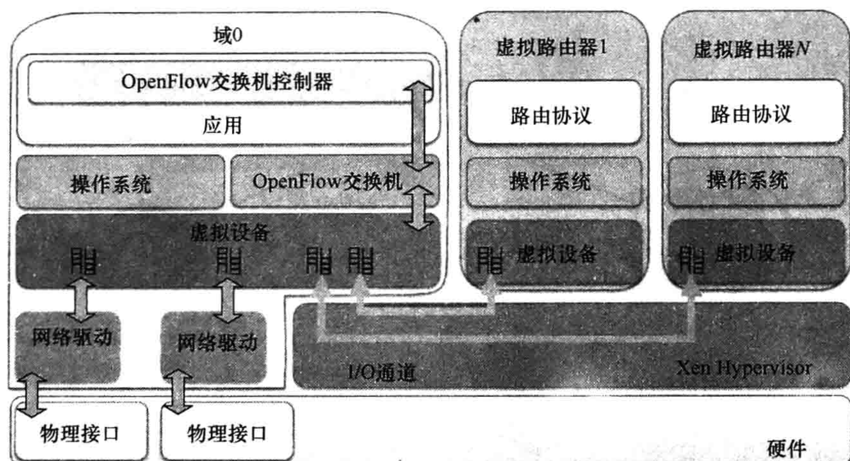


图 8.16　一个 XenFlow 网元的架构：一台 OpenFlow 交换机和一个 OpenFlow 控制器运行在物理路由器的域 0 上

理设备之间的所有通信都通过 dom0。因此，dom0 复用来自虚拟网络的报文到物理设备，并解复用来自物理设备的报文到虚拟网元［FER 11］。在 XenFlow 架构中，由一台 OpenFlow 交换机实施复用和解复用过程。依据由 OpenFlow 交换机控制器（是运行在 dom0 中的一个应用）定义的规则，对报文转发编程。这个控制器与虚拟机交互，发现由每个数据平面创建的转发规则。如果一台虚拟机（虚拟网元）要求每报文处理，则控制器将转发这个虚拟网络的所有到达流量到运行相应虚拟网元的虚拟机。除了对流编程外，这个控制器也能够为流设置策略。例如，为每条流或一组流指定一个最小的带宽［MCK 08］。

在 XenFlow 架构中，一个虚拟网络可选择与一个中心式或一个分布式控制平面一起工作。为建立采用分布式控制的一个虚拟网络，在属于那个虚拟网络的每个物理路由器（节点）中实例化一个虚拟路由器。每台虚拟路由器与 dom0 中的相应 XenFlow 控制器交互，通知转发规则。但是，在中心式控制模式中，XenFlow 的行为就像一台 OpenFlow 交换机。因此，为建立采用中心式控制的一个虚拟网络，实例化带有一个主控制器的一台中间设备。这个中央控制器与驻留在 dom0 物理路由器（寄居虚拟网元）中的个体 XenFlow 控制器交互。在这种情形中，不需要在每个物理节点中实例化一个虚拟机，这样一个模型类似于传统的 OpenFlow 网络。图 8.17给出带有三个虚拟网络的 XenFlow 架构，网络中有虚拟交换机和虚拟路由器，它们是可互操作的。使用不同虚拟网络间的 OpenFlow 协议，物理交换机数据平面是共享的。

图 8.17 给出管理实体，它们感知物理网络拓扑且由基础设施管理员管理，由他确定合适实例化或删除一个虚拟网络以及确定每个虚拟网络消耗的物理资源量。

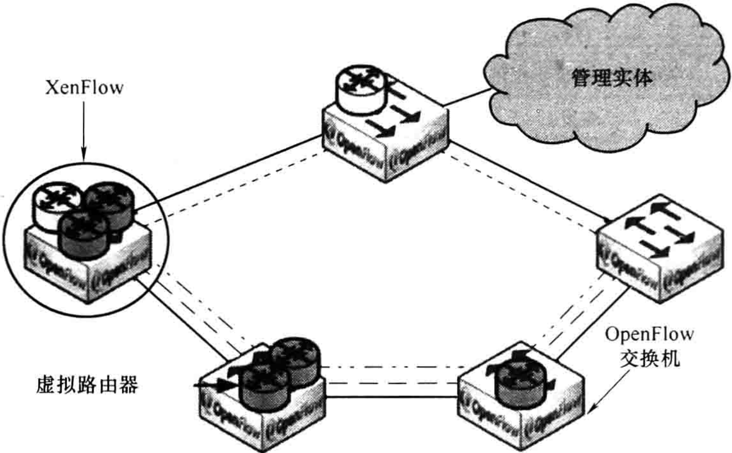


图 8.17 由虚拟交换机和虚拟路由器组成的一个 XenFlow 网络例子

这些实体也负责启动网络迁移。

1. 平面隔离和流的路由转换

当使用一台 OpenFlow 交换机时，流迁移是简单平凡的，但在 Xen 平台上路径迁移是比较复杂的，因为关闭一台路由器导致报文丢失。不过，没有报文丢失的路由器迁移是基础，且采用 XenFlow 架构可达到这个目标，这是由于采用了平面隔离技术。

每个虚拟网络的转发表由虚拟路由器计算得到，是在一台虚拟机中发生的。这个信息必须被传输到 dom0，从而可在 OpenFlow 交换机中构建正确的转发规则。图 8.18给出 XenFlow 如何实施这项任务。运行在虚拟机中的一个守护进程将数据平面信息复制到 dom0 中的一个 NOX 控制器上。之后，控制器使用规则表模块(开发作为一个 NOX 应用）将这些数据转换到流，并应需地配置 OpenFlow 交换机。

XenFlow 如下转发数据报文。到达 dom0 的一条报文被直接转发，如果它匹配流表中的任何流；否则，为了由控制器定义该报文的路径，将之转发到控制器。在这种情形中，NOX 控制器从报文中抽取 12 个 OpenFlow 字段，查询规则表，定义报文所属的网络和那个网络相应的转发规则。此后，控制器在 OpenFlow 交换机的流表中插入一条新流。重要的是指出，报文到达 dom0 时，带有虚拟路由器的目的介质访问控制（MAC）地址，且在由 OpenFlow 交换机转发之前，这个地址必须被修改为下一跳 MAC 地址。在节点是一台虚拟交换机的情形中，这项操作是不进行的。因此，这个模块确保一条虚拟链路正确地映射到一条或多条物理链路。

2. XenFlow 虚拟拓扑迁移

在一个 XenFlow 网络中，一条虚拟链路可被映射到一条或多条物理链路。报文转发是由 NOX 控制器动态编程的一个流表完成的。由此，逻辑拓扑和物理拓扑是分离的（detached)。因此，如图 8.19 所示的在一个 XenFlow 网络中的虚拟节点迁

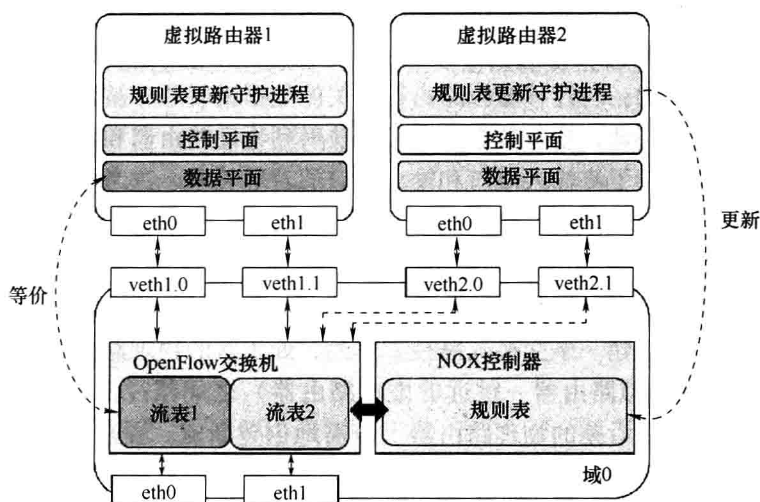


图 8.18　XenFlow 路由，其中报文直接由域 0 转发

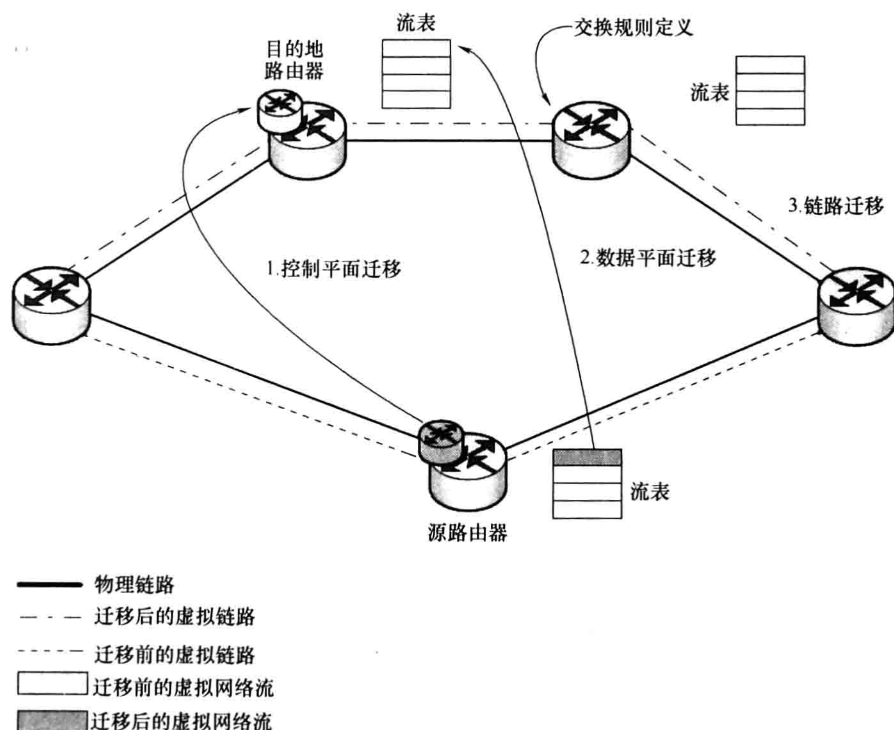


图 8.19　XenFlow 虚拟拓扑迁移的三个步骤

移，由三个步骤组成：控制平面迁移、数据平面迁移和链路迁移。以 Xen 常规实况迁移机制［CLA 05］的一种类似方式，从源到目的物理节点发生控制平面迁移。

在控制平面迁移之后，迁移虚拟网络的所有数据流必须被映射到新的虚拟拓扑。老旧的数据路径必须被转换为目的物理路由器中的一条新的数据路径。因此，数据平面迁移是这样完成的：与迁移虚拟路由器有关的数据流表项，被选中并传递到目的物理路由器；在目的地，这些数据流定义被映射到物理路由器和虚拟路由器的当前设置上。由此，保持了流输入端口和输出端口的对应关系，这考虑到在迁移源和目的地处的 dom0 虚拟交换机。之后，被迁移网络的相应流被添加到目的 dom0 OpenFlow 交换机的流表。最后，在数据平面和控制平面迁移之后，dom0 OpenFlow 交换机和其他网络交换机进行链路迁移。链路迁移在一个逻辑跳的虚拟路由器邻居和目的物理路由器之间创建一条交换式路径。由此，处于目的物理路由器和物理路由器（寄居距离被迁移虚拟路由器一跳远的虚拟路由器）之间路径上的物理路由器中，就定义了各条流。在沿路的物理路由器上应需地创建新流，要求一种自治的机制。这种机制是这样使用的，在路径中各节点的控制器规则表上引入新规则。

8.2.3 试验结果

开发了所提供架构的一个原型，作为无报文丢失的虚拟路由器迁移的概念证明。为评估性能，使用工具 Iperf [IPE 13] 作为一个报文产生器工具，使用 Tcpdump [TCP 13] 来测量所产生的、接收的和丢失的报文量。报文丢失是从所收集的信息中评估的，在负责产生和接收报文的网络接口上使用 Tcpdump。为评估 XenFlow 性能，将 XenFlow 中的迁移与 Xen 虚拟化平台的原生迁移进行比较。

试验场景由四台机器组成：两台用于报文转发，两台用于产生/接收报文。两台机器实施转发报文的功能，且 XenFlow 原型被安装在这两台机器中。每台机器装备有一个 Intel Core 2 Quad 处理器和 3 个以太网接口。这些机器运行 Xen hypervisor 4.0-amd64。在这些物理机器之一中，实例化一台虚拟机，带有一个虚拟 CPU、128MB 内存、两个网络接口和 Debian 2.6-32-5 操作系统，作为一台虚拟路由器。这些使用另外两台机器，装备有 Intel Core 2 Duo 处理器，产生或接收报文，每台机器装备有一个 1Gbit/s 以太网接口，连接到一个控制网络，装备有两个 100Mbit/s 以太网接口，与两台物理路由器通信。在这些试验中，虚拟路由器转发 64B 和 1500B 的 UDP 报文，这分别是一个以太网帧的最小和最大数据长度 [最大传输单元（MTU）]。

第一个试验测量迁移过程中控制平面下线时间。发送控制报文，报文由虚拟机转发，以便依据丢失的控制报文数来确定平均下线时间。因此，控制平面下线时间是由迁移之前直接接收到的最后一条报文的时间戳与迁移之后直接接收到的控制报文时间戳之差给定的。图 8.20 给出 XenFlow 系统和原生 Xen 迁移的控制平面下线时间，是传输报文速率的一个函数。结果表明，不管报文尺寸为何，在 XenFlow 中虚拟路由器的平均下线时间总小于 5s。但是，在原生 Xen 迁移中，虚拟路由器的平均下线时间总是较大的，范围为 12 ~ 35s。发生这个差异，主要有两个原因。首

先，在使用 XenFlow 的迁移过程中，在虚拟机内存中没有写操作，因为各报文是直接由 dom0 发送的，而在 Xen 迁移中，所有报文是由虚拟机转发的。因此，在迁移虚拟机时，在 Xen 中的报文转发产生内存写和读操作。这些写和读内存操作导致较大数量的污染页，结果，当复制虚拟机的最后一些内存页时就导致大的下线时间。其次，XenFlow 迁移是以两个步骤实施的：迁移虚拟机，之后使用 OpenFlow 迁移数据平面，这样就避免了报文丢失。在原生 Xen 迁移上，链路迁移是通过发送 ARP Reply 报文完成的，目的是表明可用的被迁移虚拟机中各个接口。但是，ARP Reply 机制是基于 ARP 表项的寿命超期的，这在更新用于与被迁移虚拟机通信的接口中增加了时延。

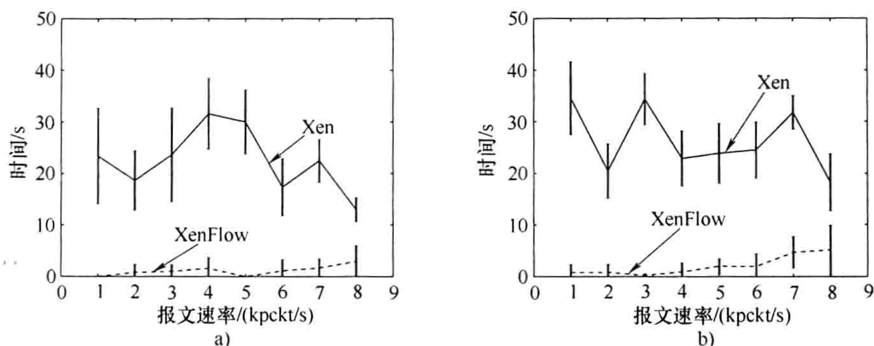


图 8.20　控制平面下线时间

a）64B 报文　b）1500B 报文

第二个试验评估总的迁移时间，它建立两个连续虚拟网元迁移之间的最小时间。总的迁移时间考虑与迁移过程有关的所有操作的执行时间。图 8.21 给出作为被传输报文速率的一个函数的总迁移时间。结果表明，相比 Xen，XenFlow 迁移要求大得多的总迁移时间。这个结果是预料中的，原因是 XenFlow 迁移涉及多个步骤，其中一个步骤是原生 Xen 迁移自己。部分额外时间的发生是由于流迁移（给定在目的物理路由器处重建数据平面的需要）和链路迁移（负责在物理网络之上建立虚拟网络的新拓扑）导致的。图 8.21b 表明，对于 1500B 大小的报文，随着传输报文速率增加，在 XenFlow 总迁移时间中有增加。这是由于如下事实导致的，即当使用 1500B 大小的报文时，在每秒大约 8000 个报文的速率下，100Mbit/s 的链路得以饱和。

第三个试验给出两个系统迁移过程中的报文丢失数。图 8.22 揭示出，在使用 XenFlow 迁移一台路由器时，没有报文丢失。随着传输报文速率增加，由 Xen 原生迁移产生的报文丢失增加，这是由于转发服务下线时间导致的丢失产生的，如图 8.20所示。因为原生 Xen 迁移下线时间几乎是常数，所以在这个时间间隔中丢失报文数随发送速率增加而增加。

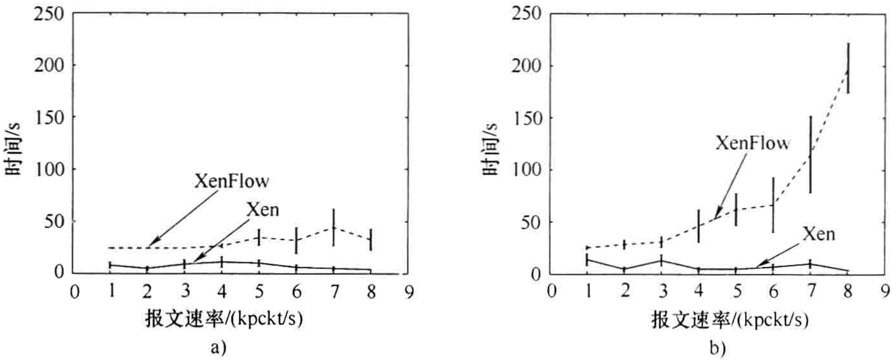


图 8.21 总迁移时间
a）64B 大小的报文 b）1500B 大小的报文

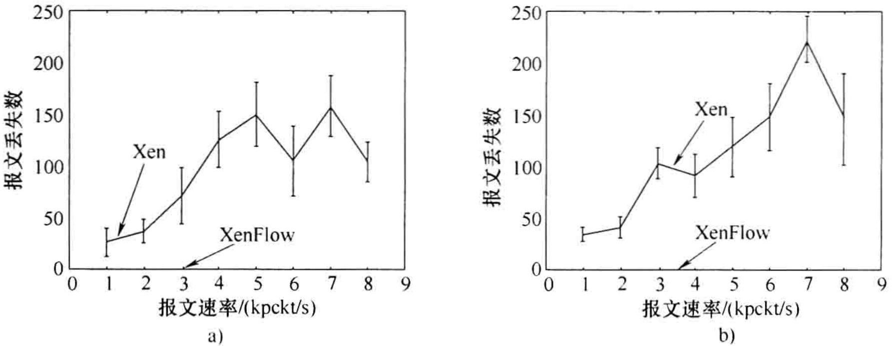


图 8.22 作为传输报文速率的一个函数的报文丢失量
a）64B 大小的报文 b）1500B 大小的报文

8.3 小结

本章给出所开发架构的一个总体视图。给出的第一个架构基于 Xen 虚拟化平台。除了新的控制和管理模块外，开发了集成一些已开发功能特征的一个模型。Xen 架构中的主要模块是资源管理器，其处理依据以前定义的策略，将物理资源分配到虚拟网络。为观测 dom0 中共享带宽、CPU 和内存的使用情况，也给出了一个模糊控制器。控制器处理虚拟机中的资源消耗，并与 Xen 调度器交互，以确保虚拟机的带宽提供。为集成这些模块，由所提出的模块和 ADAGA 修改监测功能。也添加了其他模块，如接纳控制器，它评估新的虚拟网络是否可寄居在一个特定物理节点中。接纳控制器是应用全局功能的基础，它依据网络负载重新映射虚拟网络。也为所提供架构引入了一个 QoS 模块，来完成新架构对互联网的需求。

实施了测试，评估添加到 Xen 架构中的新模块的性能。结果表明，相比参考

文献中的现有机制，所开发的机制性能较好。另外，结果也表明，所提供的接纳控制动态地将其自己适应于不同的流量需求，这保障了寄居于物理设备中虚拟网络数的准确控制。避免了资源过载和欠利用问题，这与所分析的其他建议是有区别的。也对所开发的 QoS 模块实施了测试。结果表明，与仅在虚拟网络内保障 QoS 原语相比，在虚拟网络间提供 QoS 具有优势。

开发的第二个架构使用 OpenFlow 技术和虚拟化工具，称作 OMNI。因为 OpenFlow 网络控制是中心式的，基础设施控制模块被放置在一个特殊控制器内，它与 FlowVisor 交互，控制提供给每个虚拟网络的物理资源。针对 Xen 平台开发的模块被添加，控制 OpenFlow 网络中的资源。添加到 OMNI 中的另一项功能是代理控制，它模拟分布式控制。所实施的分析表明了使用代理的优势以及管理系统对虚拟化网络的影响。

也设计了第三种方法，其重点是在一个新的虚拟化平台中 Xen 和 OpenFlow 的优势。这种方法是一种混合网络架构设计，它将 OpenFlow 交换矩阵的灵活性与基于 Xen 平台的一个网络虚拟化分布式控制组合在一起。这种方法称作 XenFlow，并实现流处理的概念，其中一个通用的网络节点能够处理任何种类的流。XenFlow 为迁移虚拟拓扑提供了一种鲁棒的和高效的方式。XenFlow 的主要目标是在虚拟路由器迁移中取得零报文丢失，并针对链路迁移，去除对隧道或外部机制的需要。该架构设计使用平面隔离技术，用作 NOX 控制器之上的一个应用，它基于转发规则控制数据平面。这些规则由运行于每台虚拟路由器中的一个守护进程加以更新。结果表明，XenFlow 控制平面下线时间要比原生 Xen 迁移下线时间低高达 30 倍。结果也表明，XenFlow 的总迁移时间要远大于原生 Xen 迁移。出现这些结果是由于 XenFlow 引入到虚拟拓扑迁移过程的新步骤，这是与 Xen 的步骤相比而言的。但是，对于虚拟路由器迁移而言，增加总的迁移时间不是一个重要因素。它仅设置两次连续迁移之间的最小时间。结果表明，XenFlow 虚拟路由器迁移是在没有报文丢失的情况下发生的，使这个架构设计适合于虚拟网络场景，这与 Xen 虚拟化平台的原生迁移的情况相反。

这些网络虚拟化工具演变为带有安全性的未来互联网测试床 [FIT 13]，该测试床被用来对新的未来互联网建议进行试验 [GUI 13]。

8.4 参考文献

[ALK 11] ALKMIM G., BATISTA D.M., DA FONSECA N.L.S., "Optimal mapping of virtual networks", *Proceedings of the IEEE Global Telecommunications Conference (IEEE Globecom 2011)*, IEEE, Houston, TX, pp. 1–6, 2011.

[ALK 12] ALKMIM G., BATISTA D.M., DA FONSECA N.L.S., "Approximated algorithms for mapping virtual networks on network substrates", *Proceedings of the IEEE International Conference on Communications (ICC 2012)*, IEEE, Ottawa, Canada, pp. 1–55, June 2012.

[BOL 09] BOLLA R., BRUSCHI R., DAVOLI F., *et al.*, "Energy-aware performance optimization for next-generation green network equipment", *Proceedings of the 2nd ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow*, ACM, pp. 49–54, 2009.

[CAR 12] CARVALHO H.E.T., FERNANDES N.C., DUARTE O.C.M.B., "SLAPv: a service level agreement enforcer for virtual networks", *Proceedings of the International Conference on Computing, Networking and Communications, Internet Services and Applications Symposium (ICNC 2012)*, Maui, Hawaii, USA, pp. 1–5, January 2012.

[CLA 05] CLARK C., FRASER K., HAND S., *et al.*, "Live migration of virtual machines", *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, USENIX Association, Anaheim, CA, EUA, pp. 273–286, April 2005.

[COU 11] COUTO R.S., CAMPISTA M.E.M., COSTA L.H.M.K., "XTC: a throughput control mechanism for Xen-based virtualized software routers", *Proceedings of the IEEE Global Communications Conference (GLOBECOM 2011)*, Houston, TX, USA, pp. 2496–2501, December 2011.

[EGI 07] EGI N., GREENHALGH A., HANDLEY M., *et al.*, "Evaluating Xen for router virtualization", *International Conference on Computer Communications and Networks (ICCCN07)*, Honolulu, Hawaii, USA, pp. 1256–1261, August 2007.

[FAJ 11] FAJJARI I., AITSAADI N., PUJOLLE G., *et al.*, "VNE-AC: virtual network embedding algorithm based on ant colony metaheuristic", *Proceedings of the ICC 2011 Next Generation Networking and Internet Symposium (ICC11 NGNI)*, IEEE, Kyoto, Japan, pp. 1–6, June 2011.

[FEA 07] FEAMSTER N., GAO L., REXFORD J., "How to lease the internet in your spare time", *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 61–64, 2007.

[FER 11] FERNANDES N., MOREIRA M., MORAES I., *et al.*, "Virtual networks: isolation, performance, and trends", *Annals of Telecommunications*, vol. 66, no. 5–6, pp. 339–355, Springer, Paris, 2011.

[FER 11a] FERNANDES N.C., MOREIRA M.D.D., DUARTE O.C.M.B., "XNet-Mon: a network monitor for securing virtual networks", *Proceedings of the IEEE International Conference on Communications (ICC 2011-Next Generation Networking and Internet Symposium)*, ICC'11 NGNI, Kyoto, Japan, 2011.

[FER 12] FERNANDES N.C., DUARTE O.C.M.B. VIPER: fine control of resource sharing in virtual networks, GTA Technical Report GTA-12-02, Electrical Engineering Program, COPPE/UFRJ, January 2012.

[FIT 13] Future Internet Testbed with Security project, available at http://www.gta.ufrj.br/fits/ (accessed in May 2013).

[GUD 08] GUDE N., KOPONEN T., PETTIT J., *et al.*, "Nox: towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[GUI 13] GUIMARÃES P.H.V., FERRAZ L.H.G., TORRES J.V., *et al.*, "Experimenting content-centric networks in the future Internet testbed environment", in *Workshop of Cloud Convergence: Challenges for Future Infrastructures and Services (WCC-02)* – ICC'2013, Budapest, Hungary, June 2013.

[IPE 13] IPERF, 2013. Available at http://iperf.sourceforge.net/ (accessed in May 2013).

[MAT 11] MATTOS D.M.F., FERNANDES N.C., DA COSTA V.T., *et al.*, (accessed in May 2013). "OMNI: OpenFlow management infrastructure", *Proceedings of the 2nd IFIP International Conference Network of the Future – NoF2011*, Paris, France, pp. 53–57, November 2011.

[MCK 08] MCKEOWN N., ANDERSON T., BALAKRISHNAN H., *et al.*, "Openflow: enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[PIS 10] PISA P., FERNANDES N., CARVALHO H., *et al.*, "Openflow and Xen-based virtual network migration", in PONT A., PUJOLLE G., RAGHAVAN S. (eds), *Communications: Wireless in Developing Countries and Networks of the Future, vol. 327 of IFIP Advances in Information and Communication Technology*, Springer, Boston, pp. 170–181, 2010.

[PIS 11] PISA P.S., COUTO R.S., CARVALHO H.E.T., *et al.*, "VNEXT: virtual network management for Xen-based testbeds", *Proceedings of the 2nd IFIP International Conference Network of the Future – NoF2011*, pp. 41–45, Paris, France, November 2011.

[RAT 05] RATNASAMY S., SHENKER S., MCCANNE S., "Towards an evolvable internet architecture", *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 313–324, 2005.

[ROD 12] RODRIGUEZ E., ALKMIM G., BATISTA D.M., *et al.*, "Green virtualized networks", *Proceedings of the IEEE International Conference on Communications (ICC 2012)*, IEEE, Ottawa, Canada, pp. 1–6, June 2012.

[SHE 09] SHERWOOD R., GIBB G., YAP K., *et al.*, Flowvisor: a network virtualization layer, Technical report, Technical Report OPENFLOW-TR-2009-01, OpenFlow Consortium, 2009.

[TCP 13] TCPDUMP & LIBPCAP, 2013. Available at http://www.tcpdump.org/ (accessed in May 2013).

[WAN 07] WANG Y., VAN DER MERWE J., REXFORD J., "VROOM: virtual routers on the move", *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, Atlanta, Georgia, USA, November 2007.

[WAN 08] WANG Y., KELLER E., BISKEBORN B., *et al.*, "Virtual routers on the move: live router migration as a networkmanagement primitive", *Proceedings of the ACM SIGCOMM*, pp. 231–242, August 2008.

[WOO 09] WOOD T., SHENOY P., VENKATARAMANI A., *et al.*, "Sandpiper: black-box and gray-box resource management for virtual machines", *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

附录　英文缩略语释义对照表

缩　略　语	全　　称	解　　释
ADAGA	Anomaly Detection for Autonomous manaGement of virtual networks	虚拟网络自治管理异常检测
AMD-V	AMD Virtualization	AMD 虚拟化
ARP	Address Resolution Protocol	地址解析协议
BGP	Border Gateway Protocol	边界网关协议
CSS	Cascading Style Sheets	级联式风格表
DMA	Direct Memory Access	直接内存访问
dom0	Domain 0	域 0
domU	User domain	用户域
DoS	Denial of Service	拒绝服务
GUI	Graphical User Interface	图形用户界面
HTTP	HyperText Transfer Protocol	超文本传输协议
ICMP	Internet Control Message Protocol	互联网控制消息协议
ISP	Internet Service Provider	互联网服务提供商
I/O	Input/Output	输入/输出
IT	Information Technology	信息技术
IVT	Intel Virtualization Technology	Intel 虚拟化技术
LAN	Local Area Network	局域网
LLDP	Link Layer Discovery Protocol	链路层发现协议
MASR	Memory Allocation, Set and Read	内存分配、设置和读取
MMU	Memory Management Unit	内存管理单元
MUC	Maximum Usage Controller	最大使用情况控制器
NIC	Network Interface Card	网络接口卡
OS	Operating System	操作系统
OSPF	Open Shortest Path First	开放最短路径优先
QoS	Quality of Service	服务质量
RIP	Routing Information Protocol	路由信息协议
RTT	Round Trip Time	往返时间
SLA	Service Level Agreement	服务水平协议

（续）

缩 略 语	全 称	解 释
SMP	Symmetric Multi Processing	对称多处理
SOAP	Simple Object Access Protocol	简单对象访问协议
SR-IOV	Single Root I/O Virtualization	单根 I/O 虚拟化
SSH	Secure Shell	安全外壳
SVG	Scalable Vector Graphics	可伸缩向量图形
TCP	Transmission Control Protocol	传输控制协议
TLV	Type-Length-Value	类型-长度-值
ToS	Type of Service	服务类型
TSC	Time Stamp Counter	时间戳计数器
UDP	User Datagram Protocol	用户数据报协议
vCPU	Virtual CPU	虚拟 CPU
VLAN	Virtual Local Area Network	虚拟局域网
VM	Virtual Machine	虚拟机
VMM	Virtual Machine Monitor	虚拟机监测器
VMS	Virtual Machine Server	虚拟机服务器
VoIP	Voice over IP	IP 上的话音（IP 电话）
VPN	Virtual Private Network	虚拟专用网
VPS	Virtual Private Server	虚拟专用服务器
XML	Extensible Markup Language	扩展标记语言
ZFG	Zero File Generator	零文件产生器

国际视野　科技前沿

国际信息工程先进技术译丛

《虚拟网络——下一代互联网的多元化方法》
《下一代融合网络理论与实践》
《认知视角下的无线传感器网络》
《移动云计算：无线、移动及社交网络中分布式资源的开发利用》
《Android系统安全与攻防》
《内容分发网络》
《计算机网络仿真OPNET实用指南》
《移动无线信道》（原书第2版）
《LTE-Advanced：面向IMT-Advanced的3GPP解决方案》
《声学成像技术及工程应用》
《认知无线电通信与组网：原理与应用》
《LTE/SAE网络部署实用指南》
《网络性能分析原理与应用》
《云连接与嵌入式传感系统》
《IP地址管理原理与实践》
《自组织网络：GSM，UMTS和LTE的自规划、自优化和自愈合》
《实现吉比特传输的60GHz无线通信技术》
《LTE自组织网络（SON）：高效的网络管理自动化》
《UMTS中的LTE：向LTE-Advanced演进》（原书第2版）
《无线传感器及执行器网络》
《UMTS中的WCDMA - HSPA演进及LTE》（原书第5版）
《认知无线电网络》
《网络融合 ——服务、应用、传输和运营支撑》
《UMTS中的LTE:基于OFDMA和SC-FDMA的无线接入》
《高性能微处理器电路设计》
《吉规模集成电路互连工艺及设计》
《高级电子封装》（原书第2版）
《基于4G系统的移动服务技术》
《移动无线传感器网 ——技术、应用和发展方向》
《UMTS蜂窝系统的Q0S与Q0E管理》
《UMTS-HSDPA系统的TCP性能》
《基于射频工程的UMTS空中接口设计与网络运行》
《未来UMTS的体系结构与业务平台：全IP的3GCDMA网络》
《环境网络：支持下一代无线业务的多域协同网络》
《基于蜂窝系统的IMS—融合电信领域的V0IP演进》
《蜂窝网络高级规划与优化 2G/2.5G/3G/ ——向4G的演进》
《微电子技术原理、设计与应用》
《多电压CMOS电路设计》
《P2P系统及其应用》
《IPTV与网络视频：拓展广播电视的应用范围》
《下一代无线系统与网络》



机械工业出版社微信服务号

WILEY

上架指导　工业技术／计算机／通信网络



ISBN 978-7-111-48726-5　定价：69.80元